A “Network Pruning Network” Approach to Deep Model Compression

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

Abstract

We present a filter pruning approach for deep model compression, using a multitask network. Our approach is based on learning a a pruner network to prune a pre-trained target network. The pruner is essentially a multitask deep neural network with binary outputs that help identify the filters from each layer of the original network that do not have any significant contribution to the model and can therefore be pruned. The pruner network has the same architecture as the original network except that it has a multitask/multi-output last layer containing binary-valued outputs (one per filter), which indicate which filters have to be pruned. The pruner's goal is to minimize the number of filters from the original network by assigning zero weights to the corresponding output feature-maps. In contrast to most of the existing methods, instead of relying on iterative pruning, our approach can prune the network (original network) in one go and, moreover, does not require specifying the degree of pruning for each layer (and can learn it instead). The compressed model produced by our approach is generic and does not need any special hardware/software support. Moreover, augmenting with other methods such as knowledge distillation, quantization, and connection pruning can increase the degree of compression for the proposed approach. We show the efficacy of our proposed approach for classification and object detection tasks.

1. Introduction

Recent advances in deep learning have led to an impressive and significant breakthroughs in various domains, such as computer vision [12, 44, 41, 9, 53, 18, 54], NLP [58, 35, 3], and information retrieval [29]. Pushing the performance further typically leads to models with overly complex, deeper architecture, which tends to increase the model size (number of parameters, depth, and breadth of layers), and FLOPs enormously, and such complex models may not be ideal to be deployed on resource-constrained devices.

This had led to considerable interest in making the model more efficient, in terms of storage as well as computation [7, 15, 50, 25, 47, 45, 33, 52, 13, 49, 48, 59]. A popular approach to increase the efficiency of the model is via model compression. Among the existing model compression approaches, the filter pruning based approaches usually show superior performance regarding FLOPs and runtime memory compression [15, 59, 7].

Selecting the most optimal subset of filters to prune from a Convolutional Neural Network (CNN) model is a combinatorially hard problem. Therefore, the existing filter pruning approaches are based on some heuristics to define filter importances. Recent works [25, 15] have shown that the strength of the feature map (output produced by a convolutional filter) dominates the output of the network. Filters with a feature map have minimal contribution to the final decision of the model; therefore, the corresponding filters can be removed from the network. In these approaches, the objective is to find the filters that are likely to produce zero (or near-zero) feature map[1]. In a pre-trained model, it is rare to get zero feature map. Therefore we optimize the network such that a majority of the filters (which are going to be pruned) have their feature map value close to zero, while the rest of the filters (that remain in the model) can still achieve accuracy close to the original network. Therefore after discarding the filters that produce zero feature-maps, the model does not incur any significant performance drop.

Most of the existing filter pruning approaches are based on heuristics to define filter importance. Defining filter importance is itself a challenging task. Also, before discarding the less important filters from the model, the representational capacity of the less important filters should be transferred to the remaining part of the model. This is a challenging task, and most of the previous approaches [27, 13, 59] exhibit poor performance in doing so and, consequently, these approaches exhibit a sharp drop in accuracy after a moderate pruning and require a high degree of finetuning, which can be very time consuming in practice.

Another drawback in the previous approaches [7, 32, 19, 23] is that they are unable to decide the layer importance. Performance of a CNN model may be very sensitive w.r.t. some of the layers, and we cannot remove a large number of filters from such layers. In contrast, some other layers

[1] A feature map is said to be zero feature map if its ℓ_1 norm is zero, i.e., all of its elements are zero.

may have a high degree of filter-level redundancy. It is a very challenging task to define layer importance precisely. Most previous methods [7, 15, 25, 13, 59] consider this as a hyperparameter (i.e., how many filters to prune from each layer). Therefore, these approaches take as input the number of filters to be pruned from each layer. To set these hyperparameters is an arduous task since, for K layers, we have K such hyperparameters. Therefore it is desirable to develop an automatic method that decides *where to prune* in the model, which motivates our approach.

We present a "network pruning network" approach for deep model compression in which we learn a *pruner network* that prunes a target (main) network. The pruner network is essentially a deep multitask network that adaptively decides which filters to prune in each layer of the target network. The objective of the multitask network is to learn weights corresponding to each output feature map of the main network (which we are going to prune) such that most of the feature maps are zero weighted without sacrificing the accuracy. Therefore the filters that correspond to zero feature maps can be safely removed from the main network without hurting the main network performance. In the proposed approach, the multitask network contains the same CNN architecture as the main network (e.g., ResNet for ResNet) but contains task-specific output layers consisting of binary outputs that denote the filters that have close to zero feature maps. The multitask network learns to maximize the number of zero feature maps in the main network. The proposed approach is end-to-end trainable using gradient descent. Our main contributions can be summarized as follows:

- The proposed approach leverages the idea of *multitask-learning*, which guides us on how to prune in each layer. We can obtain a compressed model using just a few epochs without any significant accuracy drop.
- The proposed approach uses a *multitask* network, which adaptively learns the filter importance in an end-to-end trainable manner in contrast to existing filter pruning approaches that rely on *ad hoc* heuristics to calculate the filter importances.
- Most of the existing approaches [7, 15, 25, 13, 59] require specifying *how many filters from each layer to prune* or require *a threshold* that is used to determine which filters to prune. In the proposed approach, we do not require any such input and can automatically learn the *layers importance*, thereby reducing the number of hyperparameters.

Note that, although our approach consists of two networks, i.e., a deep network to prune another deep network, it is different from student-teacher based knowledge distillation approaches [16] to deep model compression where the idea is to compress a teacher network into a simpler student network. In contrast, our approach learns a deep multitask network that prunes a target network.

2. Proposed Approach

2.1. Notation

Let us assume a CNN architecture with K convolutional layers. Assume $\mathcal{L}_i$ to be the i^{th} layer and $i \in [1, 2, \dots K]$. The layer $\mathcal{L}_i$ has n_i filters which gives the n_i feature-maps that are used as input for the next layer. The set of filters at layer $\mathcal{L}_i$ is denoted as $\mathcal{F}_{\mathcal{L}_i}$ where $\mathcal{F}_{\mathcal{L}_i} = [f_1, f_2, \dots, f_{n_i}]$. Similarly, the feature maps at layer $\mathcal{L}_i$ are represented as $\mathcal{M}_{\mathcal{L}_i} = [m_1, m_2, \dots, m_{n_i}]$. Each feature map m_i is of dimension (h_k, w_k), where h_k, w_k are height and width, respectively, of the feature map. Therefore the shape of $\mathcal{M}_{\mathcal{L}_i}$ is (h_k, w_k, n_i).

2.2. Model

This section briefly describes how the multitask network is used to prune the filters from the main network (the CNN). The core idea of our approach is to design a *multitask network* that *learns* a weight for each filter in the main network, and optimizes the main network such that most of the filters produce zero feature maps after being weighted by the multitask network. Corresponding filters in the main network that produce zero feature maps do not have any significant contribution to the model performance and can be discarded from the main network without sacrificing the model's performance.

Our approach is based on learning the weights for each filter in the main network. However, instead of associating weights to each filter, we associate weights with each feature map (the output produced by a filter of the main network). The multitask network learns these weights. The objective of the multitask network is to maximize the number of zero weights corresponding to output feature maps in the main network. The multitask network has the same architecture as the main network that we would like to prune (e.g., for pruning the ResNet main network, the multitask network is also ResNet architecture with a modified output layer). Essentially, to prune a model with K layers, we have a multitask network with K outputs, where the K outputs themselves have dimensions of size $[n_1, n_2, \dots, n_K]$. Here n_i is the number of filters at layer $\mathcal{L}_i$. We refer the *main network* as ($\mathcal{O}$) while the multitask network is called the *pruner* ($\mathcal{P}$). Fig. 1 summarizes the complete architecture of our proposed model compression framework.

Suppose the main network ($\mathcal{O}$) has a cost function $C_O(\Theta_o)$, where Θ_o denotes the parameters of the network $\mathcal{O}$. Also assume that the pruner network ($\mathcal{P}$) has a cost function $C_P(\Theta_m)$, where Θ_m denotes the parameters of the network $\mathcal{P}$. The architecture of the *pruner* is the same as the *main-network* ($\mathcal{O}$); the only difference is that the output layer is replaced by a multitask network that has K outputs (number of layers in the $\mathcal{O}$), with each of the K outputs itself being a *binary vector*. The size of the vector at layer $\mathcal{L}_i$ is n_i (size equal to the number of filters at $\mathcal{L}_i$). The complete

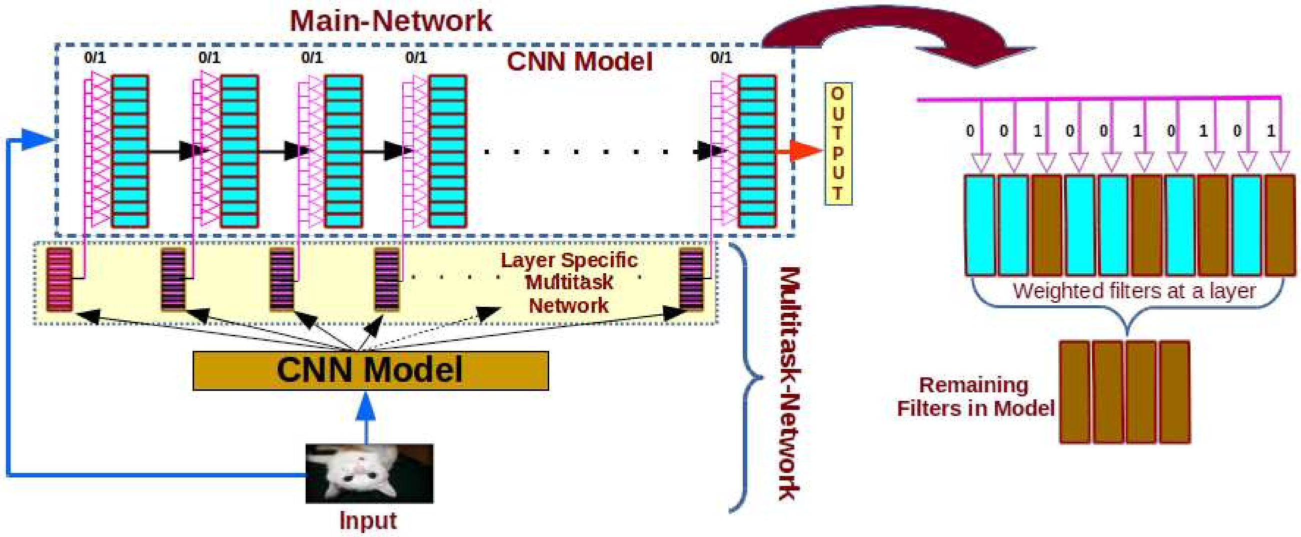

Figure 1. The upper architecture is the main network that we wish to prune, and the lower model is the same as the original one, with the only difference being that the multitask architecture replaces the output layer. Each has a task to prune a layer in the main network.

model is shown in Fig. 1.

2.3. Main-Network ($\mathcal{O}$)

The *main network* corresponds to the original network that we would like to prune. The only difference from the original network is that feature maps $\mathcal{M}_{\mathcal{L}_i}$ on each layer $\mathcal{L}_i$ are replaced by *weighted* feature maps, and the weights are given by the *pruner network* ($\mathcal{P}$) (explained in the next section). Let $\mathbf{W}_{\mathcal{L}_i} = [w_1, w_2, \dots, w_{n_i}]$ be the weights of layer $\mathcal{L}_i$ given by the $\mathcal{P}$ network. Then $\mathcal{O}$'s $\mathcal{L}_i^{th}$ layer feature maps are replaced as:

$$\mathcal{M}_w = [w_1 m_1, w_2 m_2, \dots, w_{n_i} m_{n_i}] \tag{1}$$

Here $m_1, m_2, \dots m_{n_i}$ are the feature maps at layer $\mathcal{L}_i$ in original network. Now our objective is to optimize the network with the help of $\mathcal{P}$ such that most of the w_i's are close to zero, without sacrificing the accuracy. The complete objective and joint loss are described in Section 2.5. The modified network can be easily optimized with the help of any gradient descent based optimizer.

Therefore, in the complete network, we represent each feature map as $\tilde{m}_j = w_j m_j$, here $w_j \in [0, 1]$ i.e. each feature map m_j is weighted by a weight w_j. The *pruner network* learns these weights. In the main network, $\forall w_j : w_j \rightarrow 0$ does not have any significant contribution to the overall network performance, implying that m_j can be pruned from the model.

Therefore, by discarding all the filters f_i corresponding to the $w_i \approx 0$ (feature-maps weights) from the network $\mathcal{O}$, do not significantly degrade the model's performance. Hence we can remove all such filters and corresponding feature maps from the model.

2.4. Pruner Network ($\mathcal{P}$)

The *pruner network* is the network that is responsible for the filter pruning in the main network. The *pruner network* maximizes the number of zero feature-maps in the main network. The corresponding filters that produce the zero feature-maps can be discarded from the main network. The *pruner network* give weights to each feature-maps in the main network and tries to optimize weights such that most of the $w_i \rightarrow 0$. Our *pruner network* $\mathcal{P}$ is a multitask network, with the base network being the same as the *main-network*, and the fully connected output layer replaced by multitask output layers. The number of output in multitask output layers is same as the number of layers in the model $\mathcal{O}$, i.e., K. The dimension of each multitask output is n_i (number of filters on layer $\mathcal{L}_i$). The *pruner* multitask network is shown in Fig. 2.

Let's assume that the *pruner network* has the cost function $C_P(\Theta_m)$, where Θ_m denotes its parameters. We need to optimize the model such that each of the outputs in the multitask output layer is binary, i.e., $\forall w_i : w_i \in \{0, 1\}$. To retain differentiability, we approximate the Bernoulli outputs using a scaled sigmoid on the output values. This scaled sigmoid gives a sharp change between 0 and 1. A moderate-scale value of the sigmoid can approximate the Bernoulli distribution. The scale of the sigmoid is increased gradually since experimentally we found that, if initially, we set high scale value in the network then the network is unable to learn.

Let $f(\Theta_m)$ be the output of the network $\mathcal{P}$. i.e.:

$$f(\Theta_m) = [\mathbf{W}_{\mathcal{L}_1}, \mathbf{W}_{\mathcal{L}_2}, \dots, \mathbf{W}_{\mathcal{L}_K}] \quad \forall \mathbf{W}_{\mathcal{L}_i} \in [0, 1]^{n_i} \tag{2}$$

Here $\mathbf{W}_{\mathcal{L}_i}$ denotes the i^{th} output of the multitask network and is of size n_i (number of filters on layer $\mathcal{L}_i$). In the next

Algorithm 1 Multitask Network for Model Compression

```
Require: $C_{MP}(\Theta_o, \Theta_m)$: The complete model
Require: $\alpha$ and $\beta$: learning rate and $N$: #epoch
Initialize $\Theta_o$ and $\Theta_m$ from pretrained model
while epoch$\leq$ N do
    if epoch%2==0 then
        Calculate $\Theta'_m$ by Eq:4
        i.e. $\Theta'_m \leftarrow \Theta_m - \alpha \bigtriangledown_{\Theta_m} (C^t_{MP}(\Theta_o, \Theta_m) + \lambda||f(\Theta_m)||_{l_1})$
    else
        Update $[\Theta_o, \Theta_m]$ using latest value $[\Theta_o, \Theta'_m]$ by Eq:5 i.e. $[\Theta_o, \Theta_m] \leftarrow [\Theta_o, \Theta'_m] - \beta \bigtriangledown_{[\Theta_o, \Theta'_m]} (C^{t+1}_{MP}(\Theta_o, \Theta'_m) + \lambda||f(\Theta'_m)||_{l_1})$
    end if
end while
Remove all the filters and corresponding feature maps having $w \rightarrow 0$ from the main network ($\mathcal{O}$)
Finetune the pruned model with the remaining filters
```

section, we briefly explain how we can achieve the objective of Eq. 2 without affecting the model performance. Eq. 2 gives the weights to each feature maps on every layer. The output that has the zero value gives the zero weight to the corresponding feature-map, and we can discard this feature-map and corresponding filter from the *main-network* without degrading the model performance. The objective of this network is to maximize the number of zeros in the multitask output space. The alternate optimization of the $\mathcal{O}$ and $\mathcal{P}$ ensure that the accuracy drop is minimal in the filter pruning process. In the first round, only $\mathcal{P}$ is optimized, while the parameters of $\mathcal{O}$ are kept frozen. In the second round, $\mathcal{P}$ and $\mathcal{O}$ is optimized jointly. Optimizing $\mathcal{P}$ tries to minimize the number of filters/feature-maps in the main network while optimizing $\mathcal{O}$ recovers the accuracy. Notably, our proposed approach essentially transforms the model compression problem as an end-to-end optimization problem. This can be easily optimized using stochastic gradient descent (SGD). The proposed approach automatically select the filters from each layer based on the layer importance. This fact can be easily verified since in our final compressed model's different layers have different compression rates. In contrast, other existing approaches [15, 25, 13, 59] need the number of filters to be pruned from each layer as the hyperparameters. The multitask pruner network is shown in Fig. 2.

Producing the Binary Weights: Generating the binary weights on the multitask output layer is a key point that controls the pruning rate. A high zero cardinality results in a high pruning at the cost of accuracy drop, while low zero cardinality produces a low pruning. We have to make a trade-off between the number of zeros and the accuracy drop. To produce values close to 0/1, we adopted the scaled sigmoid on the multitask output space, along with l_1 regularizer. The l_1 regularizer produces the sparsity on the output space, and sparsity can be controlled by the regularization constant. Initially, we set the scale as 1, and after a few epochs, we changed it to 30; it helps to convert the sigmoid function to nearly a step function for 0/1. The main advantage of the scaled sigmoid is that it is differentiable. The more details for each architecture are given in the experiments section.

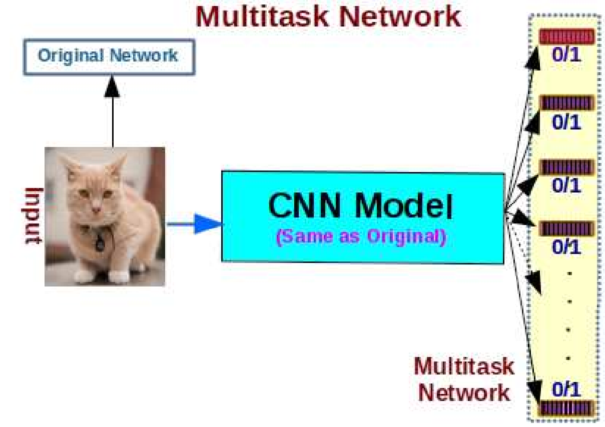

Figure 2. **Multitask pruner network:** Multitask Network has the same base model as main network but the output later is replaced by layer specific multitask network.

2.5. The Complete Model

This section explains how the complete objective is defined and the optimization is performed over the *main network* $\mathcal{O}$ and *pruner network* $\mathcal{P}$. The section also explains how the *multitask network* learns to prune the *main-network*.

Let C_{MP} be the joint loss of the *main network* and the *pruner network*, and $C_O(\Theta_o)$ and $C_M(\Theta_m)$ be the cost functions defined for $\mathcal{O}$ and $\mathcal{P}$, respectively. The joint objective can be defined as:

$$\min_{\Theta_o} \min_{\Theta_m} C_{MP}(\Theta_o, \Theta_m) + \lambda||f(\Theta_m)||_1 \quad (3)$$

Here C_{MP} is the joint loss w.r.t parameters Θ_o and Θ_m. λ is the regularization constant and $||f(\Theta_m)||_1$ is the ℓ_1 penalty on the output of the pruner network. The epoch t updates for the pruner network is given by

$$\Theta'_m \leftarrow \Theta_m - \alpha \bigtriangledown_{\Theta_m} \left(C^t_{MP}(\Theta_o, \Theta_m) + \lambda||f(\Theta_m)||_1\right) \quad (4)$$

In Eq. 4, the gradients are calculated only w.r.t. *pruner network* parameters Θ_m. The optimal parameters for pruner network Θ'_m are obtained in epoch t will be used as input in the next epoch $(t+1)$ to train the main-network. The optimization of the main network can be given as:

$$\begin{aligned}[\Theta_o, \Theta_m] \leftarrow [\Theta_o, \Theta'_m] - \beta \bigtriangledown_{[\Theta_o, \Theta'_m]} (C^{t+1}_{MP}(\Theta_o, \Theta'_m) \\ + \lambda||f(\Theta'_m)||_{l_1})\end{aligned} \quad (5)$$

Here $[\Theta_o, \Theta_m]$ denotes the joint parameters of both models and Eq. 5 uses the most recent values Θ'_m of the optimal parameters of the *pruner network*. α and β are the learning rate for Eq. 4 and 5, respectively. The optimization of Eq. 5 is performed jointly w.r.t. parameters Θ_o and Θ_m. The optimization in Eq. 4 maximizes the number of zeros in the output of the *pruner network* because of ℓ_1 penalty. At the same time, it also minimizes the loss; therefore, the model performance is maintained. Eq. 5 optimizes the model w.r.t. all parameters. Therefore, the main network has the flexibility to transfer representational capacity of the less important filters to the remaining part of the model (so as to maintain the representational capacity).

It is interesting to note that the two-step update defined by Eq. 4 and 5 is akin to the updates of a model-agnostic meta-learning (MAML) framework [8]. The only difference is that, in MAML, the optimization of *meta-learner* and *main-learner* are done over the same set of parameters. In contrast, in our proposed model compression approach, the *pruner network* parameters are a subset of the main learner's parameters. Unlike the original MAML [8] framework, there is also no "task distribution" over a dataset since here the model pruning is a single, stand-alone task that we wish to solve.

3. Related Work

Most of the recent work on model compression can be categorized into three broad categories: connection pruning, filter pruning, and quantization. The filter pruning approach has been more popular as compared to the other methods since it gives the maximum practical speedups and minimizes runtime memory, without requiring special hardware/software support. The other methods usually require special hardware/software support.

3.1. Connection Pruning

In deep CNNs, most of the weights are redundant in the model. The connection pruning is a simple method to introduce sparsity in CNN model parameters. It prunes the redundant connections from the model. One approach to compress the CNN architecture is to prune the unimportant/redundant parameters. However, it is challenging to define the importance of the parameters quantitatively. There are many approaches to rank the importance of the parameters. Optimal Brain Damage [24] and Optimal Brain Surgeon [11] used the second-order Taylor expansion to calculate the parameters importance. These approaches are based on the calculation of the second-order derivative and therefore are very costly. Blockdrop [57] proposed the skip layer approach for network compression. Using the hashing function, the method proposed in [5] randomly groups the connection weights into a single bucket and then finetunes the network to recover the performance. [10] proposes 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. The connection pruning approach is very successful when most of the parameters lie in the fully connected layer. However, these approaches result in unstructured sparsity in the model. Special hardware/software adds extra overhead. Another disadvantage of these approaches is that they are unable to save the runtime memory GPU memory.

3.2. Filter Pruning

Unlike the connection pruning approach, the filter pruning approach [17, 46, 25, 59] discards the whole filter from the model. As a result, the depth of feature maps is also reduced. The filter pruning approaches (which is the focus of our work too) do not need any special hardware or software for acceleration. Filter pruning approaches can be categorized into two categories. One class of methods find out the important filters in the model and discard the unimportant ones. After that, at each pruning step, re-training is needed to recover from the accuracy drop. [17] evaluates the importance of filters on a subset of the training data based on the output feature maps. [1] used a greedy approach for pruning. They evaluated the filter importance by checking the model accuracy after pruning the filters. [37] and [25] used a similar approach but a different metric for filter pruning. The filter pruning approach in [25] is mostly based on the weight magnitude of the filters. [6, 61, 20] used the low-rank approximation which relies on matrix factorization and can thus be costly in practice. [30] performed channel-level pruning based on the scaling factor in the training process. The pruning is done layer by layer; hence, it is very slow. The group sparsity-based approaches have also become popular for filter pruning. [22, 56, 62, 2] explored the filter pruning based on the group sparsity.

In the same vein as our work, recently [26, 14, 39] proposed automatic filter pruning approaches. [26, 39] proposed a reinforcement learning-based approach; it finds a dynamic routing path at run time to prune the model. In [14] another reinforcement learning-based model has proposed that leverage on the actor-critic model for the network pruning. These approaches use a dynamic pruning policy, while in the proposed approach, we use a single policy. Also, the proposed model was not based on reinforcement learning-based algorithms. Another popular approach are the design the efficient CNN model that can train the network from the scratch [60, 60, 51, 51]. Our work focus on the filter pruning and the efficient CNN based approach are the out the scope of this work.

3.3. Quantization

The method in [10] compressed the CNN by combining pruning, quantization, and Huffman coding. In [34], the proposed compression method was based on the floating point value quantization for model storage. These approaches assume that 32-bit float representation is redundant for the model parameters. Here we can use a lower bit configuration for model parameters without sacrificing the performance.

The extreme case of this approach can be binary bit quantization. Binarization [40] for model parameters can be used for the network compression where each floating point value is quantized to a binary value. Bayesian methods [31] have also been used for the network quantization. Most quantization methods require special hardware support to get the advantage of the compression.

4. Experiments and Results

To show the effectiveness of our proposed approach, we perform extensive experiments on large as well as small-scale datasets. We perform experiments on ResNet-50 [12] and VGG-16 [44] architecture over the large-scale dataset ImageNet [43]. We also conduct experiments on ResNet-56 [12] and VGGLike [44] architecture over the small scale dataset CIFAR-10 [21] . To show the generalization ability of our proposed approach, we also conduct an experiment over the large scale MS-COCO [28] dataset using Faster-RCNN object detector. Our experimental results demonstrate that the proposed approach yields state-of-art model compression.

4.1. Implementation Details

The proposed framework consists of two networks, *pruner network* $\mathcal{P}$ and *main network* $\mathcal{O}$. The *pruner network* $\mathcal{P}$ gives the weights to the feature maps of *main-network* $\mathcal{O}$. The objective of $\mathcal{P}$ is to maximize the number of zeros in the multitask output space, whereas $\mathcal{O}$ maintains the accuracy drop because of $\mathcal{P}$. The task given to $\mathcal{P}$ is easier than the task given to the $\mathcal{O}$. The *pruner network* can quickly maximize the number of zeros in the output space, but empirically, we find that this gives the sharp accuracy drop in the model. Since the quick optimization is irrecoverable for $\mathcal{O}$, we have to make a balance between the two networks such that the loss that occurs because of $\mathcal{P}$ can be recovered by $\mathcal{O}$. To solve this problem, we use alternating optimization; we give an equal chance to $\mathcal{O}$ to recover from the loss. Hence $\mathcal{P}$ and $\mathcal{O}$ networks are optimized by one epoch iteratively.

Our model contains binary variables. To make it differentiable, we use the scaled sigmoid $1/(1 + e^{-\alpha x})$ where α is a hyperparameter; we increase α after a few epochs once the weights produced by the output layer of $\mathcal{P}$ is uniformly distributed in [0,1]. The high α value pulls the weights close to 0/1. This helps to get the approximate Bernoulli distribution in the *pruner network* output space.

4.2. Results

4.2.1 VGG-16 on CIFAR-10

CIFAR-10 is a widely used benchmark dataset, consisting of 50000 RGB images for training and 10000 images for testing. Each image is of size 32×32. For the data augmentation, we used a horizontal flip and random crop. The VGG-16 for CIFAR-10 contains the same architecture as [44]; the only difference is that a single 512-dimensional layer is used in place of the fully connected layers. We follow the same settings as in [25]. For the base model, the network is trained for 120 epochs and has an error rate of 6.51%. The result of the proposed approach is shown in Table 1.

Method	Error%	FLOPs	Pruned Flop%
Li-pruned [25]	6.60	2.06×10^8	34.20
SparseVD [36]	7.20	–	55.95
SBP [38]	7.50	–	56.52
SBPa [38]	9.00	–	68.35
NN-1 (Ours)	**6.74**	$\mathbf{6.44 \times 10^7}$	**79.47**
NN-2 (Ours)	**7.14**	$\mathbf{5.33 \times 10^7}$	**83.00**
NN-3 (Ours)	**7.47**	$\mathbf{4.11 \times 10^7}$	**86.90**

Table 1. Pruning result on the VGG-16 over the CIFAR-10 dataset (the baseline accuracy is 93.49%).

The rate of model compression depends on the regularization constant. Three different compressed models (*NN-1, NN-2*, and *NN-3*) can be obtained by just varying the regularization constant value that controls how many zeros we want in the multitask network. We use 0.001, 0.002 and 0.005 sparse regularization constant values in the *pruner network* to obtain *NN-1, NN-2* and *NN-3* compressed models, respectively. Training of the network $\mathcal{P}$ and $\mathcal{O}$ is done in an alternating fashion. $\mathcal{P}$ tries to minimize the number of filters/feature-maps in the main network, while $\mathcal{O}$ recovers the accuracy.

Table 1 shows that the proposed approach has a high pruning rate while still maintaining accuracy. In Table 1, we can see that SBP [38] has 7.5% error on the 56.52% pruning while SBPa shows the 68.35% pruning with the 9.0% error. Our proposed approach has only 7.47% error with a high pruning rate of 86.9%.

4.2.2 ResNet-56 on CIFAR-10

Next, we experiment on ResNet-56 over the CIFAR-10 dataset. It contains three stages of convolutional layers. Each layer is connected by projection mapping and followed by the average pooling and one fully connected layer. We use the same architecture and settings as described in [25]. The same alternate optimization, as described in the previous section, is performed for maximizing the filter pruning or maximizing the zero weights produced by the *pruner network*. The network $\mathcal{P}$ is trained with the scaled sigmoid. Initially, we use scale $\alpha = 1$, and after 30 epoch, we change the scale to α=30. This new scale forces the output space of the *pruner network* to be close to 0/1. Therefore we do not have any significant accuracy drop after discarding the filters corresponding to zero weights.

Table 2 shows that the proposed approach achieves high compression rates while also giving the lowest error rate. In particular, SFP [13] has error 6.65% with the 52.6% of FLOPs pruning while the proposed approach shows the significantly better pruning 61.51% with the 6.61% error rate.

Method	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 [59]	6.99	–	43.61
CP [15]	8.20	–	50.00
SFP [13]	6.65	–	52.60
AMC [14]	8.10	–	50.00
NN-1 (Ours)	**6.61**	$\mathbf{4.85 \times 10^7}$	**61.51**

Table 2. Pruning result of ResNet-56 architecture over CIFAR-10 dataset (the baseline accuracy is 93.1%).

4.2.3 VGG-16 on ImageNet

We evaluate our approach over the large-scale ImageNet dataset [43] using the VGG-16 architecture. The same ResNet-56 alternative optimization technique is used for pruning VGG-16 networks. We train $\mathcal{P}$ network with scaled sigmoid at the output layer. We use scale $\alpha = 1$ for an initial 10 epochs, and then we set $\alpha = 30$ for the rest of the training schedules. In Table-3, we compare our result with various other pruning approaches. As shown in Table-3, our approach gives 75% FLOPs pruning with 89.71% top-5 accuracy. On the other hand, CP-4x [15] gives 75% FLOPs pruning with only 88.9% top-5 accuracy.

4.2.4 ResNet-50 on ImageNet

ResNet-50 [12] is a deep CNN architecture that has 50 layers with the residual connection. We use the same setup as proposed by the [12]. The previous approaches, such as [59, 13, 7], etc., are unable to prune the skip connection filters because of the matrix addition inconsistency. These approaches only prune the middle layer filters, resulting in limited compression. In our approach, we also prune the skip connections. To solve the addition inconsistency, we give the same weights to the output filters and the skip connection filters. Therefore it prunes the same number of the filters in the output layers and the previous skip connection layers. Hence, the proposed approach can also prune the skip connection layers. This may be very useful to prune complex networks, such as ResNet. Please refer to Figure 3 for more details.

In ResNet-50 Pruning, the *pruner* is the multitask network with the 50 tasks, because of the 50 layers in the *main-network*. We optimize the model in an alternating fashion for $\mathcal{P}$ and $\mathcal{O}$. In the first round, only $\mathcal{P}$ is optimized, while the parameters of $\mathcal{O}$ are kept frozen. In the second round, $\mathcal{P}$ and $\mathcal{O}$ is optimized jointly. The output dimension of each multitask output layer is equal to the number of filters in that layer. To get the output close to 0/1, scaled sigmoid is used. Initially, we set $\alpha = 1$ for 10 epochs, and later we use $\alpha = 50$ to get the Bernoulli weights (outputs of the multitask pruner network) on the feature maps.

Method	Acc%(Top-1)	Acc%(Top-5)	FLOPs Pruned %
Baseline	71.50	90.10	–
RNP (3X)[26]	–	87.57	66.67
ThiNet-Conv [32]	69.74	89.41	69.04
RNP (4X)[39]	–	86.67	75.00
CP 4x[15]	–	88.90	75.00
NN-1 (Ours)	**70.31**	**89.71**	**75.00**

Table 3. Pruning results for the VGG-16 over ImageNet. Our approach has minimum accuracy drop as compared to state-of-art pruning approach. We use the result reported in MatConvNet: http://www.vlfeat.org/matconvnet/pretrained/.

Method	Error%(Top-1)	Error%(Top-5)	Pruned Flop %
Baseline	24.7	7.8	-
ThiNet-70 [32]	25.97	7.9	36.8
CP [15]	–	9.2	~ 50
NISP [59]	28.0	–	44.0
SFP [13]	25.39	7.94	41.8
SPP [55]	–	9.6	~ 50
WAE [4]	–	9.6	46.8
NN-1 (Ours)	**24.58**	**7.56**	**40.7**
NN-2 (Ours)	**24.82**	**7.64**	**49.1**

Table 4. ResNet-50 Pruning results over the ImageNet dataset. The accuracy of ResNet-50 is tested using official 1-crop validation setting: center 224x224 crop from resized image with shorter side=256 (https://github.com/KaimingHe/deep-residual-networks).

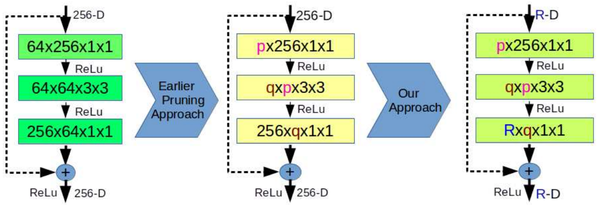

Figure 3. Unlike the previous approaches, our proposed method can also prune the skip connection filters. In the first two images, the skip connection size is fixed to 256-D, same as the original, while in the proposed approach, we also prune this to make it R-dimensional.

Empirically, we found that our approach yields compressed ResNet-50 models (NN-1, NN-2) having significantly better accuracy as compared to other approaches [59, 15, 4] because of skip connections pruning support. The proposed approach gives a significantly better pruning rate with the negligible accuracy drop. In Table 4, we show a detailed comparison with other baselines.

4.3. Generalization

To show the generalization ability of the compressed model produced by our proposed approach, we experiment on the object detection task. In this experiment, we select the popular Faster-RCNN [42] architecture on the large-scale MS-COCO [28] dataset. Our experimental results demonstrate that the compressed model produced by our proposed approach has the same generalization ability as the original model.

4.3.1 Compression for Object Detection

MS-COCO [28] is a large-scale dataset, which contains 80 object categories. The training set contains 80K images,

Model	data	Avg. Precision, IoU:		
		0.5:0.95	0.5	0.75
F-RCNN original	trainval35K	30.3	51.3	31.8
F-RCNN pruned	trainval35K	30.2	51.0	31.6

Table 5. Generalization results over MS-COCO [28] dataset for Faster-RCNN object detector. In the original Faster-RCNN, we use ResNet-50 as the base architecture while in the Faster-RCNN pruned, pruned ResNet-50 model (NN-2) from Table 4 is used. We use a publicly available implementation (https://github.com/jwyang/faster-rcnn.pytorch) for Faster R-CNN with ResNet-50 as the base network.

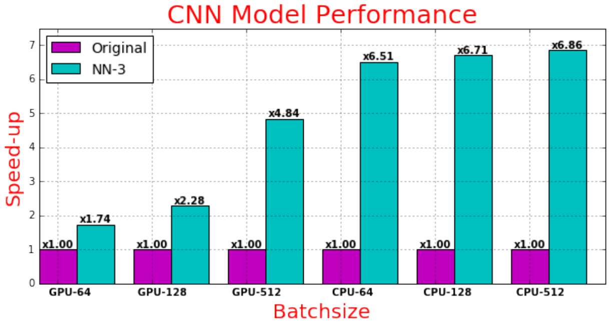

Figure 4. Practical speedup for the compressed model (NN-3) produced by the proposed approach (table-1) w.r.t. batch size on VGG-16 architecture over CIFAR-10 dataset.

and the validation set contains 35K images in total; both are combined as used as the training set called trainval35K [27]. The object detection results are reported over the 5K unused validation images (minival). The Faster-RCNN [42] is a highly popular object detection algorithm that takes the standard CNN as the base architecture for the feature extraction. For our experiments, we train the Faster-RCNN architecture with the ResNet-50 (uncompressed) [12] as the base network and the results are reported in Table 5. To show the generalization ability, we replace the base network ResNet-50 with the pruned ResNet-50 (NN-2) reported in Table 4. Repeating the same procedure of the Faster-RCNN with the pruned base model, we achieve similar results, as shown in Table 5. Therefore our compressed model not only has high FLOPs saving but also better generalization ability and can be used to higher-level computer vision tasks. In the Faster-RCNN implementation, we use ROI Align and stride 1 for the last block of the convolutional layer (layer 4) in the base ResNet-50 model.

4.4. Practical Speedup

In Fig-4, we demonstrate the practical speedup for VGG-16 compressed model (NN-3) given in the Table 1. As the Table shows, NN-3 compressed model has 7.63× theoretical FLOPs compression. We achieve $4.84\times$, $6.86\times$ practical speedup corresponding to GPU and CPU with 512 batch size. Therefore, practical CPU speedup is close to the theoretical speedup, while the GPU's practical speedup is below the theoretical speedup. This is because of the availability of thousands of cores for computation in GPU. Here one can observe that, with the increase in batch size, the parallelization ability of the model also increases; therefore, the practical speedup is close to the theoretical FLOPs compression as shown in Fig-4.

4.5. Ablation for Regularization Parameter

In Table-[1, 4] we conduct an ablation study over the λ parameter mentioned in Eq. 3. The λ parameter is used to control the pruning rate in the model. If we increase the λ value, it forces a high l_1 penalty to the multitask output vector and produces more zeros, while for lower values of λ, we get fewer zeros. These zero weight filters can be discarded from the model. In Table 1, NN1, NN2 and NN3 are compressed models for $\lambda = 0.001, 0.002$ and 0.005, respectively, and we achieve pruning rate 79.47%, 83.00% and 86.90% respectively. Similarly, in table-4, NN1 and NN2 are compressed models for $\lambda = 0.001$ and 0.002, respectively. The detail compression rate and corresponding accuracy can be seen in table [1, 4]. If we use too high pruning rate, it can dominate the model by discarding a large number of filters, and the model is unable to recover the performance.

5. Conclusion

We presented a filter pruning approach based on a multitask pruner network. The multitask network learns *where to prune* in the main network. Alternating optimization used in the proposed approach helps to achieve high FLOPs pruning rate. The multitask network tries to maximize the pruning while the main network tries to maintain accuracy during pruning. The multitask network gives approximate Bernoulli weights to each feature map in the main-network and tries to maximize the number of such zero weights. Feature maps corresponding to the zero weights produce zero-valued feature maps in the output layer; therefore, these feature maps have no contribution in the overall model. We can safely remove these feature maps with corresponding filters from the main network without degrading the model performance. One of the appealing aspects of the proposed approach is that it can automatically decide the layer importance (where to prune). The proposed approach is end-to-end without any heuristics, such as an ad-hoc specification of thresholds for filter removal. The proposed approach yields state-of-art FLOPS pruning results with minimal accuracy drop and also shows a good generalization ability 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] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

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

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

[8] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[9] C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817*, 2018.

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

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

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

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

[14] 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.

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

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

[17] 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.

[18] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. 2016.

[19] L. N. Huynh, Y. Lee, and R. K. Balan. D-pruner: Filter-based pruning method for deep convolutional neural network.

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

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

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

[23] G. Leclerc, M. Vartak, R. C. Fernandez, T. Kraska, and S. Madden. Smallify: Learning network size while training. *arXiv preprint arXiv:1806.03723*, 2018.

[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] J. Lin, Y. Rao, J. Lu, and J. Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2181–2191, 2017.

[27] 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.

[28] 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.

[29] L. Liu, F. Shen, Y. Shen, X. Liu, and L. Shao. Deep sketch hashing: Fast free-hand sketch-based image retrieval. In *Proc. CVPR*, pages 2862–2871, 2017.

[30] 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.

[31] C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. In *NIPS*, pages 3288–3298, 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] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[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] Y. Rao, J. Lu, J. Lin, and J. Zhou. Runtime network routing for efficient image classification. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[40] 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.

[41] 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.

[42] 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.

[43] 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*, pages 211–252, 2015.

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

[45] 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.

[46] 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.

[47] 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.

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

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

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

[51] 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.

[52] 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.

[53] V. K. Verma, G. Arora, A. Mishra, and P. Rai. Generalized zero-shot learning via synthesized examples. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[54] V. K. Verma, D. Brahma, and P. Rai. A meta-learning framework for generalized zero-shot learning. *AAAI*, 2020.

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

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

[57] 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.

[58] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[59] 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.

[60] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[61] 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.

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