Acceleration of Deep Convolutional Neural Networks using Adaptive Filter Pruning

Pravendra Singh *, Vinay Kumar Verma, Piyush Rai, and Vinay P. Namboodiri

***Abstract*—While convolutional neural networks (CNNs) have achieved remarkable performance on various supervised and unsupervised learning tasks, they typically consist of a massive number of parameters. This results in significant memory requirements as well as a computational burden. Consequently, there is a growing need for filter-level pruning approaches for compressing CNN based models that not only reduce the total number of parameters but reduce the overall computation as well. We present a new min-max framework for the filter-level pruning of CNNs. Our framework jointly prunes and fine-tunes CNN model parameters, with an adaptive pruning rate, while maintaining the model's predictive performance. Our framework consists of two modules: (1) An adaptive filter pruning (AFP) module, which minimizes the number of filters in the model; and (2) A pruning rate controller (PRC) module, which maximizes the accuracy during pruning. In addition, we also introduce orthogonality regularization in training of CNNs to reduce redundancy across filters of a particular layer. In the proposed approach, we prune the least important filters and, at the same time, reduce the redundancy level in the model by using orthogonality constraints during training. Moreover, unlike most previous approaches, our approach allows directly specifying the desired error tolerance instead of the pruning level. We perform extensive experiments for object classification (LeNet, VGG, MobileNet, and ResNet) and object detection (SSD, and Faster-RCNN) over benchmarked datasets such as MNIST, CIFAR, GTSDB, ImageNet, and MS-COCO. We also present several ablation studies to validate the proposed approach. Our compressed models can be deployed at run-time, without requiring any special libraries or hardware. Our approach reduces the number of parameters of VGG-16 by an impressive factor of 17.5X, and the number of FLOPS by 6.43X, with no loss of accuracy, significantly outperforming other state-of-the-art filter pruning methods.**

***Index Terms*—Deep convolutional neural network acceleration, Pruning, Model compression, Efficient computation.**

I. Introduction

DEEP convolutional neural networks (CNN) have been used extensively for object recognition and various other computer vision tasks. After the early works based on standard forms of deep convolutional neural networks [1], [2], recent works have proposed and investigated various architectural changes [3]–[5] to improve the performance of CNNs. Although these changes, such as adding more layers to the CNN or increasing number of convolutional filters per layer, have led to impressive performance gains, they have also resulted in a substantial increase in the number of parameters, as well as the computational cost. The increase in model size and computations have made it infeasible to deploy these models on embedded and mobile devices for real-world applications. To address this, recent efforts have focused on several approaches for compressing CNNs, such as using binary or quantized [6] weights. However, these require specialized hardware.

The approaches based on pruning of unimportant/redundant weights [7]–[10] give limited speedup. As most of the CNN parameters reside in the fully connected layers, a high compression rate with respect to the number of network parameters can be achieved by simply pruning redundant neurons from the fully connected layers. However, this does not typically result in any significant reduction in computations (FLOPs based speedup), as most of the computations are performed in convolutional layers. For example, in the case of VGG-16, the fully connected layers contain 90% of total parameters but account for only 1% of computations, which means that convolutional layers despite having about 10% of the total parameters are responsible for 99% of computations. This has led to a considerable recent interest in convolutional layer filter pruning approaches. However, most existing pruning approaches [7], [8] result in irregular sparsity in the convolutional filters, which requires software specifically designed for sparse tensors to achieve speedups in practice [7]. In contrast, some other filter pruning approaches [11]–[13] are designed to directly reduce the feature map width by removing specific convolutional filters via ℓ_2 or ℓ_1 regularization on the filters, and effectively reducing the computation, memory, and the number of model parameters. These methods result in models that can be directly used without requiring any sparse libraries or special hardware.

In this work, we propose a novel filter pruning formulation. Our formulation is based on a simple min-max game between two modules to achieve an adaptive maximum pruning with minimal accuracy drop. We show that our approach results in substantially improved performance as compared to other recently proposed filter pruning strategies while being highly stable and efficient to train. We refer to the two modules of our framework as an Adaptive Filter Pruning (AFP) and Pruning Rate Controller (PRC). The AFP is responsible for pruning the convolutional filter, while the PRC is responsible for maintaining accuracy. In this way, we are iteratively pruning the least important filters from the deep CNNs. We also enforce orthogonality of weights (orthogonality regularization) in training deep convolutional neural networks to reduce

* Corresponding author.

Part of this work has been accepted in preliminary form in the 28th International Joint Conference on Artificial Intelligence (IJCAI-2019). Our IJCAI paper "Play and Prune: Adaptive Filter Pruning for Deep Model Compression" can be found on this link: https://www.ijcai.org/proceedings/2019/480.

The authors are with the Department of Computer Science and Engineering, IIT Kanpur, Uttar Pradesh, 208016, India (e-mail: psingh@cse.iitk.ac.in; vkverma@cse.iitk.ac.in; piyush@cse.iitk.ac.in; vinaypn@cse.iitk.ac.in).

Manuscript received MMMM, YYYY; revised XXXX, YYYY

IEEE Journal of Selected Topics in Signal Processing: Special Issue on Compact Deep Neural Networks with Industrial Applications.

redundancy across filters.

Unlike most previous approaches, our approach does not require any *external* fine-tuning. In each epoch, it performs an adaptive fine-tuning to recover from the accuracy loss caused by the previous epoch's pruning. By external fine-tuning, we refer to dedicated epochs of fine-tuning for the final compressed model. Our approach jointly prunes and fine-tunes the CNN model parameters, with an adaptive pruning rate, while maintaining the model's predictive performance. Therefore, in a given epoch, we can recover from the accuracy drop suffered due to the previous epoch's pruning, making the model ready to prune filters in the current epoch. Therefore, our approach does internal fine-tuning in the same epoch. Moreover, while previous approaches need to pre-specify a pruning level for each layer, our approach is more flexible in the sense that it directly specifies an error tolerance level and, based on that, decides which filters to prune, and from which layer(s).

Through an extensive set of experiments and ablation studies on several benchmarks, we show that the proposed approach provides state-of-the-art filter pruning, and significantly outperforms existing methods. In particular, our approach significantly reduces the computations, memory requirements, model size, and the number of parameters. Our approach reduces the number of parameters of VGG-16 by an impressive factor of 17.5X, and the number of FLOPS by 6.43X, with no loss of accuracy.

Our key contributions are as follows:

- Simplicity and efficiency of the approach, which is based on a simple yet principled min-max iterative two-player game.
- Unlike earlier pruning approaches, our pruning technique has an adaptive pruning rate. The PRC module controls the pruning rate dynamically while considering layer importance. The PRC module also ensures that the compressed model will not go beyond the error tolerance limit (iterative pruning bounds accuracy drop).
- We prune the least important filters and, at the same time, reduce the redundancy level in the deep model by using orthogonality constraints during training.
- The proposed approach has been evaluated for various popular models (LeNet, VGG, MobileNet, ResNet, SSD, and Faster-RCNN) over benchmarked datasets (MNIST, CIFAR, GTSDB, ImageNet, and MS-COCO).

II. Related Work

Among one of the earliest efforts on compressing CNNs by pruning unimportant/redundant weights, [6], [14] includes binarizing/quantizing the network weights, which reduces the computation time as well as storage requirement. The disadvantage of this approach is that it requires specialized hardware to run the deployed model. Transfer learning-based methods have also been used for model compression. One such approach is by [15], [16], which transfers/distills the knowledge of a massive-sized network to a much smaller network. Another popular approach is to use a *sparsity* constraints on the neuron's weights. These approaches [8], [17] learn sparse network weights, where most of the weights are zero, and consequently can lead to very small model sizes. However, despite the sparsity of the weights, the pruned models are still not computationally efficient at runtime. Moreover, these models require special library/hardware for sparse matrix multiplication because activation/feature maps are still dense, which hinders practical utility.

A very initial approach proposed an interesting method [18] that prunes exactly one node by solving a system of linear equations in each pruning iteration. However, such methods are practically impossible to apply on the deep CNNs as the computations would be infeasible. More recently, the work in [19] proposes a scaling factor to scale the outputs of various structures for model compression. Further, the method in [20] uses normalized cross-correlation between all filter pairs to enforce the diversity between filters. We differ from this approach [20] as we have used the spectral norm in orthogonality constraint to enforce diversity between filters. This is known to be more optimal [21]. In our work, we use this more optimal orthogonality constraint and pose it in a play and prune architecture to obtain a more diverse set of filters.

The work in [22] proposes a structured sparsity learning(SSL) approach to regularize the structures of deep CNNs to reduce computation cost. This approach introduces structured sparsity in the model by group Lasso regularization during the training. Similarly, the work in [23] propose a Bayesian model that considers the computational structure of deep CNNs and provides structured sparsity. This approach injects noise to the outputs of the neurons while keeping the weights unregularized by removing neurons with a low SNR from the computation graph. The work in [24] uses a sparse variational dropout technique to sparsify deep CNNs.

Most of the popular approaches that focus on model compression are based on sparsifying the fully connected layers since, typically, about 90% of the network parameters are in the fully connected layers. However, note that the bulk of the computations take place in the convolutional layers, and consequently, these approaches do not result in computational acceleration. Only a few recent works have had the same focus as our work, i.e., on filter pruning [11]–[13], [25], [26], that can be practically useful. In [12], the authors proposed filter pruning by ranking filters based on their sum of absolute weights. They assumed that if the sum of absolute weights is sufficiently small, the corresponding activation map will be weak.

The work in [26] suggests an iterative approach to transfer the representational capacity of its convolutional layers to a fraction of the filters and then prune the redundant ones followed by a re-training step to restore the accuracy. Similarly, [27] use a different approach to rank the filter importance, based on the entropy measures. The assumption is that high entropy filters are more important. Alternatively, [25] uses a data-driven approach to calculate filter importance, which is based on the average percentage of zeros in the corresponding activation map. Less important filters have more number of zeros in their activation map. Recently, [28] proposed improving run time by using a Taylor approximation. This approach estimates the change in cost by pruning the filters. Another work [11] uses pruning of filters based on the next layer statistics. Their approach is based on checking the activation map of the next layer to prune the convolution filters from the current layer. In a recent work [13] used a similar approach as in [11] but used lasso regression.

Runtime Neural Pruning (RNP) [29] proposes a framework that prunes the deep CNNs dynamically at the runtime by performing pruning in a bottom-up, layer-by-layer manner. SFP [30] enables the pruned filters to be updated when training the deep CNNs after pruning. Neuron Importance Score Propagation (NISP) [31] calculates the importance scores of final responses to every neuron in the deep CNNs. The deep model is pruned by removing neurons with the least importance, followed by re-training steps to recover accuracy drop.

Unlike the aforementioned pruning approaches, our pruning approach has an adaptive pruning rate. The PRC module controls the pruning rate dynamically while considering layer importance. We prune the least important filters and, at the same time, reduce the redundancy level in the model by adding orthogonality constraints during training.

III. Proposed approach

We assume we have a CNN model $\mathcal{M}$ with K convolutional layer. Layer i is denoted as $\mathcal{L}_i$ and consists of n_i filters denoted as $F_{\mathcal{L}_i} = \{f_1.f_2.\dots,f_{n_i}\}$. We assume that the unpruned model $\mathcal{M}$ has the accuracy of $\mathcal{E}$ and, post-pruning, the error tolerance limit is ϵ.

A. Overview

Our deep model compression framework is modeled as a min-max game between two modules, Adaptive Filter Pruning (AFP) and Pruning Rate Controller (PRC). The objective of the AFP is to iteratively minimize the number of filters in the model, while PRC iteratively tries to maximize the accuracy with the set of filters retained by AFP. The AFP will prune the filter only when the accuracy *drop* is within the tolerance limit (ϵ). If accuracy drop is more than ϵ then pruning stops, and the PRC tries to recover the accuracy by fine-tuning the model. If PRC's fine-tuning is unable to bring the error within the tolerance level ϵ, the AFP will not prune the filter from the model and game converges.

Let us denote the AFP by $\mathcal{P}$ and the PRC by $\mathcal{C}$. Our objective function can be defined as follows:

$$\max_{\#w} \mathcal{C}\left(\min_{\#w=\sum_{i=1}^{K} n_i} \mathcal{P}\left(F_{\mathcal{L}_1}, F_{\mathcal{L}_2}, \dots F_{\mathcal{L}_K}\right)\right) \tag{1}$$

As shown in the above objective, the AFP ($\mathcal{P}$) minimizes the number of filters in the network, and the PRC ($\mathcal{C}$) optimizes the accuracy given that the number of filters. Here $\#w$ is the number of remaining filters after pruning by AFP.

An especially appealing aspect of our approach is that the pruning rates in each iteration are decided *adaptively* based on the performance of the model. After each pruning step, the controller $\mathcal{C}$ checks the accuracy drop (see Fig. 1). If the accuracy drop is more than ϵ, then the pruning rate is reset to zero, and the controller $\mathcal{C}$ tries to recover the system performance (further details of this part are provided in the section on PRC). Eq. 1 converges when $\mathcal{C}(\#w)$ performance drop is more than the tolerance limit, and it is unable to recover it. In such a case, we rollback the current pruning and restore the previous model. At this point, we conclude that this is an optimal model that has the maximal filter pruning within ϵ accuracy drop.

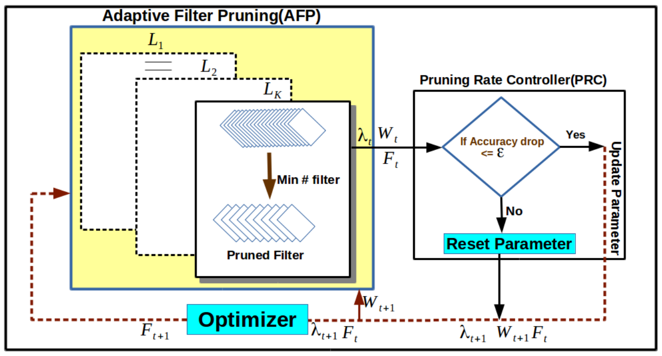

Fig. 1. The figure shows the complete architecture. Here AFP minimizes the number of filter in model while PRC maximizes the accuracy during pruning. Here $\lambda_t, \mathbf{W_t}$ and $\mathbf{F_t}$ are the regularization parameter, weight-threshold and remaining filters in the model respectively at t^{th} pruning iteration.

B. Convolutional Filter Partitioning

The pruning module $\mathcal{P}$ first needs to identify a candidate set of filters to be pruned. For this, we use a filter partitioning scheme in each epoch. Suppose the entire set of filters of the model $\mathcal{M}$ is partitioned into two sets, one of which contains the important filters while the other contains the unimportant filters. Let $\mathcal{U}$ and $\mathcal{I}$ be the set of unimportant and important filters, respectively, where

$$\mathcal{M} = \mathcal{U} \cup \mathcal{I} \quad \text{and} \quad \mathcal{U} \cap \mathcal{I} = \emptyset \tag{2}$$

$$\mathcal{U} = \{U_{\mathcal{L}_1}, U_{\mathcal{L}_2}, \dots, U_{\mathcal{L}_K}\} \text{ and } \mathcal{I} = \{I_{\mathcal{L}_1}, I_{\mathcal{L}_2}, \dots, I_{\mathcal{L}_K}\}$$

Here $U_{\mathcal{L}_i}$ and $I_{\mathcal{L}_i}$ are set of unimportant and important filters, respectively, in layer $\mathcal{L}_i$. $U_{\mathcal{L}_i}$, selected as follows:

$$U_{\mathcal{L}_i} = \underset{\text{top } \alpha\%}{\sigma} \left(\text{sort}(\{|f_1|, |f_2|, \dots, |f_{n_i}|\})\right) \tag{3}$$

Eq. 3 sorts the set in increasing order of $|f_j|$, σ is the select operator and selects the $\alpha\%$ filters with least importance. The remaining filters on $\mathcal{L}_i$ belongs to set $I_{\mathcal{L}_i}$. Here $|f_j|$ is the sum of absolute values of weights in convolutional filter f_j and can be seen as the filter importance. A small sum of absolute values of filter coefficients implies less importance. Our approach to calculate filter importance uses their ℓ_1 norm (Eq. 3), which has been well-analyzed and used in prior works [12], [26], [30]. Our approach isn't however tied to this criterion, and other criteria can be used, too. We are using this criterion because of its simplicity.

C. Weight Threshold Initialization

After obtaining the two sets of filters $\mathcal{U}$ and $\mathcal{I}$, directly removing $\mathcal{U}$ may result in a sharp and potentially irrecoverable accuracy drop. Therefore we treat $\mathcal{U}$ as a candidate set of filters

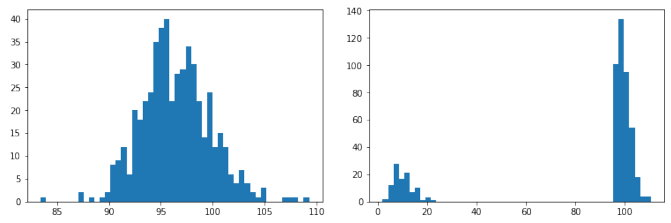

Fig. 2. Histogram of the sum of absolute value of convolutional filters for CONV5_1 in VGG-16 on CIFAR-10. Where left plot is for original filters and the right plot shows the sum of the absolute value of filters after optimization.

to be pruned, of which a *subset* will be pruned eventually. To this end, we optimize the original cost function for the CNN, subject to a group sparse penalty on the set of filters in $\mathcal{U}$, as shown in Eq. 4. Let $C(\Theta)$ be the original cost function, with Θ being original model parameters. The new objective function can be defined as:

$$\Theta = \arg\min_{\Theta} \left(C(\Theta) + \lambda_A ||\mathcal{U}||_1\right) \tag{4}$$

Here λ_A is the ℓ_1 regularization constant. This optimization penalizes $\mathcal{U}$ such that $|f_j|$ (sum of absolute weights of coefficients in each filter f_j) tends to zero, where $f_j \in U_{\mathcal{L}_i}$ $\forall i \in \{1, 2, \ldots, K\}$. This optimization also helps to transfer the information from $\mathcal{U}$ to the rest of the model. If a filter f_j has approximately zero sum of absolute weights then it is deemed safe to be pruned. However, reaching a close-to-zero sum of absolute weights for the whole filter may require several epochs. We therefore choose an *adaptive* weight threshold (W_{γ_i}) for each layer $\mathcal{L}_i$, such that removing $\forall f_j \in \mathcal{U}_{\mathcal{L}_i}$s.t.$|f_j| \leq W_{\gamma_i}$ results in negligible (close to 0) accuracy drop.

We calculate the initial weight threshold (W_{γ_i}) for $\mathcal{L}_i$ as follows: optimize Eq. 4 for one epoch with $\lambda_A = \lambda$, where λ is the initial regularization constant, which creates two clusters of filters (if we take the sum of the absolute value of filters) as shown in Fig. 2. On the left cluster (right plot) using the binary search find the maximum threshold W_{γ_i} for $\mathcal{L}_i$ such that accuracy drop is nearly zero.

D. Adaptive Filter Pruning (AFP)

The objective of the AFP module is to minimize the number of filters in the model. Initially, based on the sparsity hyperparameter λ, we calculate the weight thresholds $\mathbf{W}$. Now instead of using the constant pruning rate, we change the pruning rate adaptively given by the pruning rate controller (PRC; described in the next section) in each epoch. This adaptive strategy helps to discard the filter in a balanced way, such that we can recover from the accuracy drop. In each epoch, from the current model, we select $\alpha\%$ of the filter of lowest importance from each layer, partition them into $\mathcal{U}$ and $\mathcal{I}$, and perform optimization using Eq. 4, where λ_A is given by PRC. The optimization in Eq. 4 transfers the knowledge of unimportant filters into the rest of the network. Therefore some filter from the $\mathcal{U}$ can be safely discarded. This removal of the filter from the model is done based on the threshold ($\mathbf{W_A}$) given by the PRC module. Now, from each layer, the filters below the adaptive threshold $\mathbf{W_A}$ are discarded. In each epoch, the weight thresholds and regularization constant is updated dynamically by the PRC module, and a subset of $\mathcal{U}$ is pruned. Hence, in the same epoch, we can recover from the accuracy drop from the previous epoch's pruning, making the model ready to prune filters in the current epoch.

The objective of the AFP module can be defined as:

$$\Theta' = \underset{\#w \in \Theta'}{\sigma} \left[\mathcal{P}\left(\arg\min_{\Theta'} \left(C(\Theta') + \lambda_A \sum_{i=1}^{K} ||\mathcal{U}||_1 \right) \right) \right] \tag{5}$$

Here Θ' is the collection of remaining filters after pruning, and σ is the select operator. $\#w$ is the collection of all the filters from each layer $\mathcal{L}_i$ that has a sum of absolute value greater than the W_{γ_i}. From Eq.-5, it is clear that it minimizes the number of the filters based on $W_{\gamma_i} \in \mathbf{W_A}, \forall i \in \{1, 2, \ldots, K\}$.

E. Pruning Rate Controller (PRC)

Let $\mathbf{W} = [W_{\gamma_1}, W_{\gamma_2}, \ldots, W_{\gamma_K}]$ denote the initial weight thresholds for the K layers (described in Weight Threshold Initialization section). Now the adaptive thresholds $\mathbf{W_A}$ are calculated as follows:

$$\mathbf{W_A} = \delta_w \times T_r \times \mathbf{W} \tag{6}$$

$$T_r = \begin{cases} \mathcal{C}(\#w) - (\mathcal{E} - \epsilon) & : \mathcal{C}(\#w) - (\mathcal{E} - \epsilon) > 0 \\ 0 & : Otherwise \end{cases} \tag{7}$$

where $\mathcal{C}(\#w)$ is the accuracy with $\#w$ remaining filters, $\mathcal{E}$ is the accuracy of the unpruned network, and the number $\mathcal{C}(\#w) - (\mathcal{E} - \epsilon)$ denotes how far we are from tolerance error level ϵ. Here, δ_w is a constant used to accelerate or decrease the pruning rate. The regularization constant λ_A in Eq. 4 also adapted based on the model performance after pruning and its updates are given as follows

$$\lambda_{\mathbf{A}} = \begin{cases} (\mathcal{C}(\#w) - (\mathcal{E} - \epsilon)) \times \lambda & : \mathcal{C}(\#w) - (\mathcal{E} - \epsilon) > 0 \\ 0 & : Otherwise \end{cases} \tag{8}$$

Form Eq. 8 it is clear that we set the regularizer constant to zero if our pruned model performance is below the tolerance limit. Otherwise, it is proportional to the accuracy above the tolerance limit. λ is the initial regularization constant.

The PRC module essentially controls the rate at which the filters will get pruned. In our experiments, we found that if the pruning rate is high, there is a sharp drop in accuracy after pruning, which may or may not be recoverable. Therefore pruning saturates early, and we are unable to get the high pruning rate. Also, if the pruning rate is too slow, the model may get pruned very rarely and spends most of its time in fine-tuning. We, therefore, use a pruning strategy that adapts the pruning rate *dynamically*. In the pruning process, if in some epoch, the system performance is below the tolerance limit, we reset the pruning rate to zero. Therefore the optimization will focus only on the accuracy gain until the accuracy is recovered to be again within the tolerance level ϵ. Note that the adaptive pruning rate depends on model performance. When the model performance is within ϵ, the pruning depends on how far we are from ϵ. From Eq 6, it is clear that the $\mathbf{W_A}$

depends on the performance of the system over the $\#w$ filters in the model. In this way, by controlling the pruning rate, we maintain a balance between filter pruning and accuracy. This module tries to maximize accuracy by reducing the pruning rate. The objective function of PRC can be defined as:

$$\max_{\Theta'} \mathcal{C}\left(\Theta', D\right) \tag{9}$$

Here $\mathcal{C}$ calculates the performance, i.e., accuracy. It is the function of all the convolutional filters Θ' that remain after pruning, and D is the validation set used to compute the model accuracy.

In addition to dynamically controlling the pruning rate, the PRC offers several other benefits, discussed next.

1) Iterative Pruning Bounds Accuracy Drop: Eq. 7 and Eq. 8 ensure that the compressed model will not go beyond the error tolerance limit, which is controlled by the PRC. Experimentally we found that, in a non-iterative one round pruning, if the model suffers from a high accuracy drop during pruning, then pruning saturates early, and fine-tuning will not recover accuracy drop properly. We have shown an ablation study that shows the effectiveness of iterative pruning over the single round pruning to justify this fact.

2) Pruning Cost: The cost/effort involved in pruning is mostly neglected in most of the existing filter pruning methods. Moreover, most of the methods perform pruning and fine-tuning separately. In contrast, we jointly prune and fine-tune the CNN model parameters, with an adaptive pruning rate, while maintaining the model's predictive performance. Therefore, in a given epoch, we can recover from the accuracy drop suffered due to the previous epoch's pruning and making the model ready to prune filters in the current epoch.

In addition, the adaptive pruning rate has the following other cost/effort advantages over fixed pruning rate:

- If the pruning rate is too high, then it will result in sharp accuracy drop after pruning, which may require several epochs to recover. It may also be possible that the accuracy drop is irrecoverable. In such a case, most methods rollback the current state (extra overhead) and again start from the previous state. Hence more epochs (high pruning cost) are required to get the final compressed model.
- If the pruning rate is too low, the model may get pruned very slowly, therefore, again high pruning cost.

In our approach, an adaptive pruning rate is controlled by PRC (Eq. 6,7,8). On ImageNet, we get compressed model ResNet-50 PP-2 after 62 epochs (without any rollback). The experimental section shows more details.

3) Layer Importance: Most previous methods [12], [13], [26], [30], [31] use user-specified desired model compression rate but finding optimal compression rate is not so easy and involves many trials. In CNNs, some layers are relatively less important, and therefore we can prune many more filters from such layers. In contrast, if we prune a large number of filters from important layers, then this might result in an irrecoverable loss in accuracy. Our approach is more flexible since it directly specifies an error tolerance level ϵ and, based on that, adaptively decides which filters to prune, and from which layer(s), using Eq. 6 to determine layer-specific pruning rates.

IV. Orthogonality Constraint in Pruning

In our proposed pruning approach described in the previous section, we prune the less important filters from the model. These discarded filters do not have any significant contribution to the model. Let us consider a possibility where two filters are important and, therefore, not discarded in the pruning process. However, they might share a high degree of similar information (high correlation). Therefore, redundancy may also be possible in a compressed model.

To address this issue, we enforce orthogonality across filters to encourage filter diversity. To this end, we augment our pruning method with Orthogonality Constraint (OC) to ensure filter diversity. We add an orthogonality regularization during training time to make filters orthogonal/uncorrelated, which results in improved performance of the deep CNNs. Orthogonality Constraint (OC) can be applied on the convolutional layers as well as fully connected layers.

The deep model has a set of convolutional layers, followed by fully connected layers. We have to apply orthogonality constraint in each layer (fully connected as well as convolutional layers). Let us assume that the deep model contains K convolutional layers and F_C fully connected layers. The weights in i^{th} fully-connected layer can be denoted as $X_i \in \mathbb{R}^{m_i \times n_i}$. The weights in j^{th} convolutional layer can be denoted as $X_j \in \mathbb{R}^{w \times h \times i \times o}$, where w, h, i, and o are filter width, filter height, number of input and output channels respectively. We can reshape X_j into a matrix form $X'_j \in \mathbb{R}^{m' \times n'}$, where $m' = w \times h \times i$ and $n' = o$.

Using the above notation, we can define orthogonality constraint for the convolutional layers, as shown below:

$$\mathcal{O}_c = \delta \sum_{j=1}^{K} \sigma\left(X'_j {X'}_j^T - I\right) \tag{10}$$

Similarly, orthogonality constraint for the fully connected layers can be given as:

$$\mathcal{O}_f = \delta \sum_{i=1}^{F_C} \sigma\left(X_i X_i^T - I\right) \tag{11}$$

In the above, $\sigma(M_t)$ denotes the spectral norm of M_t, i.e., the largest singular value of M_t and δ is the weight given to orthogonality constraint [21]. Therefore, the overall orthogonality-based regularization term is given by

$$\mathcal{O} = \mathcal{O}_c + \mathcal{O}_f \tag{12}$$

This regularizer $\mathcal{O}$ (Eq. 12) is added to Eq. 4, and Eq. 5 to ensure filter diversity. The orthogonality constraint reduces redundancy across filters which results in improved performance as shown in experimental section.

V. Experiments and Results

To show the effectiveness of the proposed approach, we have conducted extensive experiments on small as well as large datasets (MNIST [32], GTSDB [33], CIFAR-10 [34], ILSVRC-2012 [35], and MS-COCO [36]). Our approach yields state-of-art results on LeNet [1], VGG [3], MobileNet

TABLE I
PRUNING RESULTS FOR THE LENET-5 ARCHITECTURE ON MNIST.

Method	Filter	Error%	FLOPs	Pruned Flop %
SSL-2 [22]	5,19	0.80	5.97×10^5	86.42
SSL-3 [22]	3,12	1.00	2.89×10^5	93.42
SBP [23]	–	0.86	–	90.47
SparseVD [24]	–	0.75	–	54.34
PP-1 (Ours)	**4,5**	**0.95**	$\mathbf{1.95 \times 10^5}$	**95.56**
PP-OC-1 (Ours)	**4,5**	**0.80**	$\mathbf{1.95 \times 10^5}$	**95.56**

TABLE II
LAYER-WISE PRUNING RESULTS AND PRUNED MODELS (PP-1, AND PP-2) STATISTICS FOR VGG-16 ON CIFAR-10.

		Baseline	**PP-1**	**PP-2**
Input Size		**32×32×3**	**32×32×3**	**32×32×3**
Layers	CONV1_1	64	18	18
	CONV1_2	64	48	48
	CONV2_1	128	65	65
	CONV2_2	128	65	65
	CONV3_1	256	104	96
	CONV3_2	256	112	112
	CONV3_3	256	114	110
	CONV4_1	512	207	186
	CONV4_2	512	163	79
	CONV4_3	512	79	79
	CONV5_1	512	74	74
	CONV5_2	512	48	48
	CONV5_3	512	60	60
	FC6	512	512	512
	FC7	10	10	10
Total parameters		15.0M	1.13M (13.3×)	0.86M (17.5×)
Model Size		60.0 MB	4.6 MB (13.0×)	3.5 MB (17.1×)
Accuracy		93.49	93.46	93.35
FLOPs		313.7M	54.0M (5.8×)	48.8M (6.43×)

[37], SSD [38], Faster RCNN [39], and ResNet [5] respectively. For all our experiments, we set $\lambda = 0.0005$ (set initially but later adapted), $\delta_w = 1$ and $\alpha = 10\%$. We follow the same parameter settings and training schedule as [12], [13], [30]. We also report an ablation study for various values of α. We perform experiments without/with Orthogonality Constraint (OC) and observe that adding Orthogonality Constraint (OC) in the training process improves the CNNs performance. We use the same settings and setups as described in [21] for Orthogonality Constraint (OC).

A. LeNet-5 on MNIST

MNIST (Modified National Institute of Standards and Technology) is a handwritten digits dataset. It contains 60,000 training images and 10,000 test images. Our LeNet-5 architecture contains two convolutional layers and two fully connected layers. The complete architecture is 20-50-800-500. We trained LeNet-5 on MNIST from scratch and achieved 0.83% error rate.

We achieve a higher FLOPs compression rate as compared to the previous approaches (Table I). In prior work, SSL-3 [22] reports an error of 1.0% on 93.42% FLOPs pruning while we achieve 95.56% FLOPs pruning with 0.80% error (PP-OC-1). Table I shows a detailed comparison with previous approaches. PP-1 denotes the compressed model without using Orthogonality Constraint (OC). PP-OC-1 denotes the compressed model when using Orthogonality Constraint (OC) in the pruning process. PP-OC-1 shows better performance as compare to PP-1 because the former promotes filter diversity.

TABLE III
COMPARISON OF PRUNING VGG-16 ON CIFAR-10 (THE BASELINE ACCURACY IS 93.49%).

Method	Error(%)	Params Pruned(%)	Pruned FLOPs(%)
Li-pruned [12]	6.60	64.0	34.20
SBP [23]	7.50	–	56.52
AFP-E [26]	7.06	93.3	79.69
AFP-F [26]	7.13	93.5	81.39
PP-1 (Ours)	**6.54**	92.5	**82.8**
PP-OC-1 (Ours)	**6.40**	92.5	**82.8**
PP-2 (Ours)	**6.65**	**94.3**	**84.5**
PP-OC-2 (Ours)	**6.57**	**94.3**	**84.5**

B. VGG-16 on CIFAR-10

We experimented with VGG-16 architecture on the CIFAR-10 dataset. We follow the same parameter settings and training schedule as [12]. Input size is $32 \times 32 \times 3$ for VGG-16 on CIFAR-10. It contains one fully connected layer of size 512 after convolutional layers, and the rest of the architecture is same as [3].

We collected $\alpha = 10\%$ filters from all convolutional layers to make $\mathcal{U}$ in each epoch. For weight threshold initialization, we optimize Eq. 4, as described in the proposed approach. This optimization partition the filters into two clusters. Please refer to Fig. 2 to see the behavior of the optimization. This polarization helps to easily decide the filters to be pruned from the $\mathcal{U}$ because now the separation between $\mathcal{U}$ (left cluster) and $\mathcal{I}$ (right cluster) is relatively high. Please note that we applied ℓ_1-regularization only on $\alpha\%$ filters, but the behavior (sum of absolute weights) of the remaining $(100 - \alpha\%)$ filters are also changed to adapt the representational capacity of its convolutional layers.

From Fig. 2, it is clear that for left cluster $|f_i| \in [1, 21]$ while for the right cluster $|f_i| \geq 95$. Because of this polarization, we can safely remove the subset of filters from $\mathcal{U}$ below the threshold given by PRC. Our first pruned model is VGG-16 PP-1. Table II shows the layer-wise pruning statistics for PP-1 (first pruned model), and PP-2 (second pruned model). We compare our results with the recent works on filter pruning. Our approach consistently performs better as compared to Li-pruned [12], SBP [23], AFP [26] as shown in Table III. PP-OC-2 shows better performance as compare to PP-2 because of the presence of filter diversity.

C. Ablation Study for VGG-16 on CIFAR-10

This section shows a detailed analysis of the effect on the different components in the proposed approach.

1) Ablation Study on the hyper-parameter α: We did an ablation study on the hyper-parameter α, i.e., how many filters are selected for partitioned $\mathcal{U}$. We experimented with $\alpha = 5, 10, 20, 30\%$. We found that if we take the lower value, it will not degrade the performance (Table IV) but takes more epochs to prune. While if we take high α (say 30%) value, it starts degrading the performance of the model early, hence we are unable to get high pruning rate. In our case, we find $\alpha = 10\%$ is a moderate value, and at this rate, we can achieve a high pruning rate. This rate we set across all architecture like LeNet, VGG, ResNet, and MobileNet.

TABLE IV
ABLATION STUDY OVER THE α VALUES. EXPERIMENTALLY WE FOUND THAT $\alpha = 10$ IS THE MOST SUITABLE.

Alpha value	Error(%)	Parameters	FLOPs
5	6.54	1.12×10^6	5.3×10^7
10	6.54	1.13×10^6	5.4×10^7
20	6.56	1.15×10^6	5.5×10^7
30	6.61	1.27×10^6	6.1×10^7

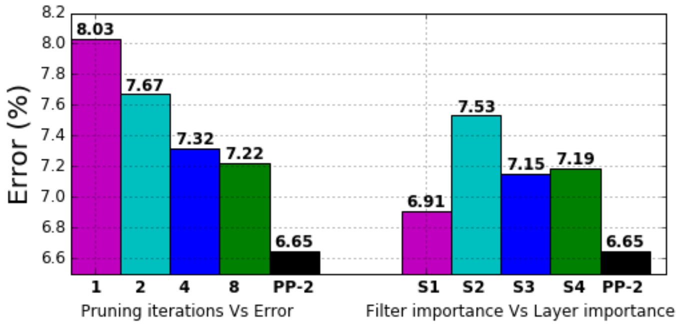

Fig. 3. (a) Left figure shows the effectiveness of iterative pruning (b) Right figure shows effect of layer importance on error for the same FLOPs pruning (84.5%)

2) *Pruning iterations Vs Error:* In Fig. 3 (left), we have shown that if we do the pruning in 1 shot, it has significant accuracy drop (8.03%) as compared to PP-2 on the same FLOPs pruning (84.5%). While if we prune the model iteratively, we have less error rate for the same FLOPs pruning. Therefore, if we do pruning manually without using PRC (pruning iterations: 1, 2, 4, 8), the error rate is always worse than our pruned model PP-2 which uses PRC.

3) *Filter importance Vs. Layer importance:* Most of the previous approach [12], [26], [31] focus on the *How to prune* (filter importance/ranking), while it is also important for the better pruning rate to decide *Where to prune*. Our approach also decides the *where to prune* by considering layer importance. In the Fig. 3 (right), we are showing the ablation on our approach's capability to decide the importance of the layers. In Fig. 3 (right), we are showing the error rate on the similar FLOPs pruning without/with considering layer importance for the four compressed model search (S1,S2,S3,S4) with our approach (PP-2) using same filter importance criteria. The compressed models (S1,S2,S3,S4) do not use PRC, which is why to have error more than our compressed model PP-2.

4) *Training compressed model with randomly initialized weights:* In this, we are analyzing the effect of fine-tuning on the compressed model rather than training from scratch. We train two compressed models VGG-16 PP-1 and VGG-16 PP-2 from scratch with the same architecture as in Table-II. Unfortunately, the errors are 7.23%, 7.36% respectively, which are much worse than our pruned models as shown in Table-V. Similar observation on CIFAR-10 is also observed in [12]. Hence instead of fine-tuning the compressed model, if we directly perform training on the compressed model with randomly initialized weights (training from scratch), then the model may suffer from accuracy drop as shown in Table-V. The key reason behind this can be the involvement of a highly non-convex optimization problem, for which a certain degree of parameter redundancy is required during training. But after training, we can remove these redundant parameters without much effect on the performance of the model.

TABLE V
TABLE SHOWS THE PARAMETERS PRUNED (PP), AND FLOPS PRUNED (FP) RESULTS FOR VGG-16 ON THE CIFAR-10 IN DIFFERENT SETUPS.

Model	Error%	Method	PP(%)	FP(%)
Baseline	6.51	–	–	–
PP-1 (ours)	**6.54**	pretrained model used	92.5	82.8
PP-2 (ours)	**6.65**	pretrained model used	94.3	84.5
PP-1	7.23	training from scratch	92.5	82.8
PP-2	7.36	training from scratch	94.3	84.5

TABLE VI
TABLE SHOWS THE PARAMETERS PRUNED (PP), AND FLOPS PRUNED (FP) RESULTS FOR VGG-16 ON THE CIFAR-10 UNDER DIFFERENT REGULARIZATIONS.

Model	Error%	Method	PP(%)	FP(%)
PP-1 (ours)	6.54	ℓ_1 regularization	92.5	82.8
PP-2 (ours)	6.65	ℓ_1 regularization	94.3	84.5
PP-1	6.55	ℓ_2 regularization	92.5	82.8
PP-2	6.63	ℓ_2 regularization	94.3	84.5

TABLE VII
COMPARISON OF PRUNING RESNET-56 ON CIFAR-10 (THE BASELINE ACCURACY IS 93.1%).

Method	Error(%)	Pruned FLOPs(%)
Li-B [12]	6.94	27.6
NISP [31]	6.99	43.6
CP [13]	8.20	50.0
SFP [30]	6.65	52.6
AFP-G [26]	7.06	60.9
PP-1 (Ours)	**6.91**	**68.4**
PP-OC-1 (Ours)	**6.85**	**68.4**

5) *Use of ℓ_2 regularization in place of ℓ_1 regularization in Eq. 4 and Eq. 5:* The performance of ℓ_1 regularization and ℓ_2 regularization is similar as shown in Table VI. Note that we have applied regularization only on $\mathcal{U}$ part, and we are pruning the whole convolutional filter from the model. The role of regularization in our approach is to reduce the sum of absolute weights of filters belonging to $\mathcal{U}$ part. Therefore ℓ_1 regularization or ℓ_2 regularization can be used to reduce the sum of absolute weights of filters belonging to $\mathcal{U}$ part.

D. ResNet-56 on CIFAR-10

We follow the same parameter settings and training schedule as [12], [30]. Our approach significantly outperforms various state-of-the-art approaches for ResNet-56 on CIFAR-10. The results are shown in Table VII. We achieve a high pruning 68.4% with the 6.85% error rate, while AFP-G [26] has the error rate of 7.06% with 60.9% pruning.

E. MobileNet on CIFAR-10

We train MobileNetV2 [37] on the CIFAR-10 dataset and achieve 91% accuracy because of three downsample layers, leading to 4×4 feature maps before the last avg pooling. Therefore, we changed the number of downsampling layers from three to two and achieved 94.5% accuracy. We use the publicly available code[1] for these experiments, which uses two

[1]https://github.com/tinyalpha/mobileNet-v2_cifar10

TABLE VIII
PRUNING RESULTS FOR MOBILENET ON CIFAR-10 DATASET.

Method	Error(%)	Pruned FLOPs(%)
Baseline [37]	5.5	–
PP-1 (Ours)	5.7	**43.6**
PP-OC-1 (Ours)	**5.5**	**43.6**

TABLE IX
COMPARISON OF PRUNING VGG-16 ON IMAGENET (THE BASELINE ACCURACY IS 90.1%).

Method	Top-5 Accu.(%)	Pruned FLOPs(%)
RNP (3X) [29]	87.57	66.67
ThiNet-70 [11]	89.53	69.04
DDSSS [19]	88.20	75.24
SLAF [20]	89.38	77.85
CP [13]	88.20	80.00
PP-1 (Ours)	**89.81**	**80.20**
PP-OC-1 (Ours)	**90.01**	**80.20**

TABLE X
COMPARISON OF PRUNING RESNET-50 ON IMAGENET (THE BASELINE ACCURACY IS 92.2%).

Method	Top-5 Accu.(%)	Parameters	Pruned FLOPs(%)
ThiNet [11]	90.7	16.94M	36.9
SFP [30]	92.0	–	41.8
PP-1 (Ours)	**92.0**	**15.7M**	**44.1**
PP-OC-1 (Ours)	**92.1**	**15.7M**	**44.1**
CP [13]	90.8	$\sim$ 18M	50.0
PP-2 (Ours)	**91.4**	**13.7M**	**52.2**
PP-OC-2 (Ours)	**92.0**	**13.7M**	**52.2**

downsampled layers.

MobileNet [37], [40] is a highly compact model. Therefore, accelerating an already compact model is a challenging task. We take the original MobileNetV2 model as the baseline. We prune 43.6% FLOPs in MobileNet with no loss in accuracy. Experimental results show that we can accelerate MobileNetV2 using the proposed approach. The results are shown in the table-VIII.

F. VGG-16 On ILSVRC-2012

To show the effectiveness of our proposed approach, we also experimented with the large-scale dataset ILSVRC-2012 [35]. It contains 1000 classes with 1.5 million images. To make our validation set, randomly 10 images (from the training set) are selected from each class. This is used by PRC to calculate the validation accuracy drop for adjusting the pruning rate. In this experiment, α is the same as the previous experiment. We follow the same setup and settings as [13]. The baseline[2] accuracy is 90.1%.

Our large-scale experiment for VGG-16 [3] on the ImageNet [35] shows the state-of-art result over the other approaches for model compression. Channel-pruning (CP) [13] has the 80.0% model FLOPs compression with the top-5 accuracy 88.2%, while we have same FLOPs pruning (80.2%) with the top-5 accuracy 90.01%. Refer to table-IX for the detail comparison results. Our compressed model (PP-1) is obtained after 38 epochs. PP-OC-1 denotes the compressed model with using Orthogonality Constraint (OC) in the pruning process.

[2]http://www.vlfeat.org/matconvnet/pretrained/

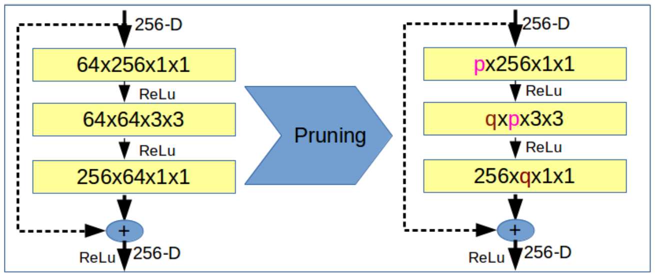

Fig. 4. Our ResNet pruning strategy, where we pruned first two convolutional layers in each block.

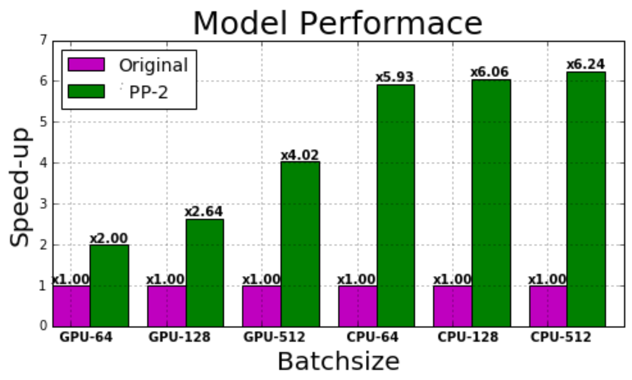

Fig. 5. Speedup corresponding to CPU (i7-4770 CPU@3.40GHz) and GPU (GTX-1080) over the different batch size for VGG-16 on CIFAR-10.

G. ResNet-50 ON ILSVRC-2012

In ResNet, there exist restrictions on the few layers due to its identity mapping (skip connection). Since for $output = f(x) + x$ we need to sum two vector, therefore we need x and $f(x)$ should have same dimension. Hence we cannot change the output dimension freely. Hence two convolutional layers can be pruned for each block (see Fig. 4). Unlike the previous work [11], where they explicitly set $p = q$, we have not imposed any such restriction, which results in more compression with better accuracy. We prune ResNet-50 from block 2a to 5c continuously as described in the proposed approach. If the filter is pruned, then the corresponding channels in the batch-normalization layer and all dependencies to that filter are also removed. We follow the same settings as [13]. The baseline[3] accuracy is 92.2%.

Our results on ResNet are shown in Table X. We are iteratively pruning convolutional filters in each epoch as described earlier. PP-1 is obtained after 34 epochs. Similarly, PP-2 is obtained after 62 epochs. We have experimentally shown that our approach reduces FLOPs and Parameters without any significant drop in accuracy.

H. Practical Speedup

The practical speedup is sometimes very different by the result reported in terms of FLOPs prune percentage. The practical speedup depends on the many other factors, for

[3]https://github.com/KaimingHe/deep-residual-networks

TABLE XI
CLASS WISE AP FOR SSD512-ORIGINAL AND SSD512-PRUNED MODEL ON GTSDB DATASET.

Model	**AP**				**Size**	**Parameters**
	prohibitory	**mandatory**	**danger**	**mAP**		
SSD512-Original	96.8	86.9	87.1	90.3	98.7 MB	24.7M
SSD512-Pruned	97.5	87.6	88.0	**91.0**	**2.9 MB (34.0×)**	**0.73M (3.0%)**

TABLE XII
GENERALIZATION RESULTS ON MS-COCO. PRUNED RESNET-50 (PP-2) USED AS A BASE MODEL FOR FASTER-RCNN.

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	30.3	51.1	31.7

example, intermediate layers bottleneck, availability of data (batch size), and the number of CPU/GPU cores available.

For VGG-16 architecture with the 512 batch size, we have 4.02X practical GPU speedup, while the theoretical speedup is 6.43X (Fig. 5). This gap is very close on the CPU, and our approach gives the 6.24X practical CPU speedup compared to 6.43X theoretical (Fig. 5).

I. Generalization Ability

We also conduct experiments on object detection network to show the generalization ability of compressed models produced by our approach. We select two most popular object detector SSD [38] on GTSDB dataset and Faster RCNN [39] on MS-COCO [36] dataset. In SSD experiment, we get $\sim$34$\times$ compression in terms of model parameters with significant improvement in AP. We use ResNet-50 as a base network in Faster RCNN implementation.

1) SSD-512 on German traffic detection benchmarks: In this section, we test the generalization ability of our pruned model (VGG-16 PP-2), which is pruned on CIFAR-10. In the first experiment, we trained original SSD-512 (Single Shot MultiBox Detector) on German traffic detection benchmarks (GTSDB) [33] dataset. For this experiment, ImageNet pre-trained base network (VGG-16) is used. In the second experiment, we substitute the base network of SSD-512 with our pruned VGG-16 PP-2 model. SSD detects objects in real-time at multiple scales from multiple layers. Usually, initial layers detect the smaller object, and last layers detect the bigger object. After training, we notice that the model is over-fitted. Object sizes in the GTSDB dataset are very small. Hence, SSD‘s later layer feature maps are unable to capture the objects, which result in over-fitting in the model by later layers. Therefore our pruned SSD-512 model detects the object from the first detection (CONV4_3) layer. We remove all layers after CONV4_3 in the pruned SSD-512 model. We achieve a significant mAP improvement and $\sim$34$\times$ compression in model size after removing later detection layers. Table XI shows the detailed experimental results.

2) Faster RCNN on COCO: The experiments are performed on COCO detection datasets with 80 object categories [36]. Here all 80k train images and a 35k val images are used for training (trainval35K) [41]. We are reporting 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-XII.

In this experiment, we used our pruned ResNet-50 model (PP-2) as given in Table-X as a base network in Faster-RCNN. We found that the pruned model shows similar performances in all cases. In the Faster-RCNN implementation, we use ROI Align and use stride 1 for the last block of the convolutional layer (layer4) in the base network. We have used a publicly available code[4] for Faster R-CNN with ResNet-50 as a base network. In Faster-RCNN, we change the base model with our pruned ResNet-50 (PP-2) model to get the compressed Faster R-CNN. From our results, we show that the Faster-RCNN based on ResNet-50 gives similar results with our pruned ResNet-50 model. Therefore, our pruned ResNet-50 model retains its feature representation capacity even after pruning.

VI. CONCLUSION

We proposed an Adaptive Filter Pruning framework to accelerate Deep Convolutional Neural Networks. Our approach follows a min-max game between two modules (AFP and PRC). Since our approach can prune the entire convolution filter, there is a significant reduction in FLOPs and the number of model parameters. Unlike earlier pruning approaches, our pruning technique has an adaptive pruning rate. The PRC module controls the pruning rate dynamically while considering layer importance. The PRC module also ensures that the compressed model will not go beyond the error tolerance limit. Our framework prunes the least important filters and, at the same time, reduces the redundancy level in the model by using orthogonality constraints during training. Our approach does not require any special hardware/software support, is generic, and practically usable. We have performed extensive evaluations of our approach for various popular models (LeNet, VGG, MobileNet, ResNet, SSD, and Faster-RCNN) over benchmarked datasets (MNIST, CIFAR, GTSDB, ImageNet, and MS-COCO). Our approach can also be used in conjunction with pruning methods such as binary/quantized weights, weight pruning, etc. These can directly be applied to the pruned model given by our method to get further speedups and model compression. The experimental results show that our proposed framework achieves state-of-art results on standard architectures and generalizes well for the object detection task.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[4]https://github.com/jwyang/faster-rcnn.pytorch

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *NIPS*, 2012, pp. 1097–1105.

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

[4] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *CVPR*, 2017, pp. 1251–1258.

[5] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016, pp. 770–778.

[6] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *ECCV*. Springer, 2016, pp. 525–542.

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

[8] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *NIPS*, 2015, pp. 1135–1143.

[9] J. M. Alvarez and M. Salzmann, “Learning the number of neurons in deep networks,” in *NIPS*, 2016, pp. 2270–2278.

[10] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in *NIPS*, 2016, pp. 2074–2082.

[11] J.-H. Luo, J. Wu, and W. Lin, “Thinet: A filter level pruning method for deep neural network compression,” *ICCV*, 2017.

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

[13] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in *ICCV*, vol. 2, 2017, p. 6.

[14] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, “Compressing neural networks with the hashing trick,” in *ICML*, 2015, pp. 2285–2294.

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

[16] J. Ba and R. Caruana, “Do deep nets really need to be deep?” in *NIPS*, 2014, pp. 2654–2662.

[17] H. Zhou, J. M. Alvarez, and F. Porikli, “Less is more: Towards compact cnns,” in *ECCV*. Springer, 2016, pp. 662–677.

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

[19] Z. Huang and N. Wang, “Data-driven sparse structure selection for deep neural networks,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 304–320.

[20] T. Wang, L. Fan, and H. Wang, “Simultaneously learning architectures and features of deep neural networks,” *ICANN*, 2019.

[21] N. Bansal, X. Chen, and Z. Wang, “Can we gain more from orthogonality regularizations in training deep networks?” in *Advances in Neural Information Processing Systems*, 2018, pp. 4261–4271.

[22] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in *NIPS*, 2016, pp. 2074–2082.

[23] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, “Structured bayesian pruning via log-normal multiplicative noise,” in *NIPS*, 2017, pp. 6778–6787.

[24] D. Molchanov, A. Ashukha, and D. Vetrov, “Variational dropout sparsifies deep neural networks,” in *ICML*, 2017, pp. 2498–2507.

[25] 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.

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

[27] J.-H. Luo and J. Wu, “An entropy-based pruning method for cnn compression,” *arXiv preprint arXiv:1706.05791*, 2017.

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

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

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

[31] 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.

[32] Y. LeCun, “The mnist database of handwritten digits,” *http://yann. lecun. com/exdb/mnist/*.

[33] 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*, no. 1288, 2013.

[34] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” 2009.

[35] 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*, vol. 115, no. 3, pp. 211–252, 2015.

[36] 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*. Springer, 2014, pp. 740–755.

[37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[38] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *ECCV*, 2016, pp. 21–37.

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

[40] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[41] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, “Feature pyramid networks for object detection.” in *CVPR*, 2017, p. 4.

Pravendra Singh Biography text here.

Vinay Kumar Verma Biography text here.

Piyush Rai Biography text here.

Vinay P. Namboodiri Biography text here.