Feature Independent Filter Pruning by Successive Layer Analysis

Milton Mondal*§, Bishshoy Das*§
Pushpendra Singh†, Brejesh Lall*, Sumantra Dutta Roy*, Shiv Dutt Joshi*
* Department of Electrical Engineering, Indian Institute of Technology Delhi
† Department of Electronics & Communication Engineering, National Institute of Technology Hamirpur

***Abstract*—Convolutional neural networks (CNN) have become deeper and wider over time to represent the dataset accurately. However, due to low computational power and storage, mobile devices or embedded systems cannot use deeper models. Filter pruning solves this by eliminating redundant filters from the model while maintaining performance. Pruning can happen in a data-dependent or independent manner. Data-dependent methods require extensive computations because they use feature map values that change for each training example, whereas data-independent methods only use fixed filter weights. This paper aims to design a data-free filter pruning algorithm that can match or surpass the performance of data-dependent methods. Existing data-independent methods use only present layer filter properties to determine the filter importance, which is a major problem. Our analysis suggests for the first time in this paper that both the present and next layer filter information are crucial to determine the filter importance as it minimizes the feature map reconstruction error of the next layer. We propose a novel method named 'Filter Removal by Analyzing only Network's Kernels (FRANK)', which prunes filters adaptively from different convolutional layers. FRANK is neither data-driven nor does it use any additional network or module, but extensive experiments with different datasets (CIFAR, ImageNet) and architectures (VGG, ResNet, MobileNet) demonstrate its effectiveness over state-of-the-art (SOTA) methods. FRANK reduces half of the computational burden of VGG16 but improves CIFAR10 and CIFAR100 classification accuracy. Even for larger datasets like ImageNet, FRANK achieves 42.7% FLOPs reduction while maintaining top-1 accuracy for ResNet50.**

***Index Terms*—Data Independent Filter Pruning, Convolutional Neural Networks, Deep Learning, Adaptive Pruning, Model Compression**

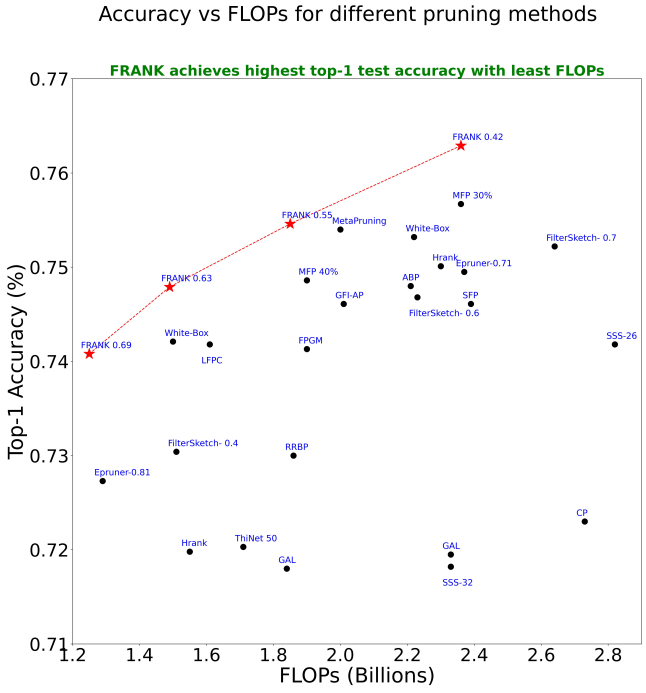

Fig. 1. Top-1 accuracy vs FLOPs for different pruning methods for ResNet50 while using ILSVRC-2012 dataset. FRANK provides the best test accuracy compared to state of the art (SOTA) methods

I. Introduction

Convolutional neural networks (CNN) have been extremely successful in solving diverse computer vision tasks, which include image classification, object detection, image captioning, etc. The advancement of computational and storage resources over the years has made this achievable because we need deeper and larger models for better performance. Currently, workstations and GPU servers are able to use CNN as these devices are capable of storing millions of parameters and performing billions of floating point operations (FLOPs). For instance, the ResNet50 model requires 25.56 million parameters and approximately 4 billion FLOPs to process a single image of the ImageNet dataset. However, the high computational cost of using CNN restricts its deployment in mobile devices or embedded systems due to resource constraints. To deal with this, researchers have proposed several model compression methods and efficient training mechanisms which can produce a compact model without degrading the performance significantly. These solutions include (a) parameter quantization, (b) tensor decomposition, (c) knowledge distillation, (d) compact network synthesis, (e) pruning network parameters, etc. Out of all these solutions, pruning network parameters reduces the computational cost of deep neural networks (DNN) by first identifying and then deleting the redundant parameters in such a manner that the learning effectiveness is maintained or even increased over the unpruned (/base) model. Unlike network synthesis methods, pruning offers the flexibility to the user to choose a standard base model (like VGG [1], ResNet [2], MobileNet [3], etc.), and thereafter it automatically generates a compact model from the base model by trimming the

§Equal contribution

redundant parameters.

Researchers propose mainly two kinds of parameter pruning methods, (i)weight pruning and (ii) filter pruning. Weight pruning methods delete the unimportant weights from a model. So, even if a few weights of a convolutional kernel are found to be redundant by the weight pruning method, then it prunes those weights. Therefore, we achieve a sparse model using weight pruning, but the model will have an irregular structure. So, the obtained pruned model would fail to use Basic Linear Algebra Subprograms (BLAS) libraries directly. In contrast, a filter pruning method produces a compact model with no structural irregularity as it eliminates a filter entirely. Thus, filter pruning is preferred over weight pruning as we do not require any additional specialized hardware or software to deal with structural issues. So, our focus in this paper is to design a less complex yet effective 'filter pruning' algorithm that determines and removes the redundant filters efficiently.

There are two types of filter pruning methods (i) Data Dependent [4]–[10], (ii) Data Independent [11]–[17]. The methods which use feature map information to perform pruning are termed as 'Data Dependent' methods whereas the methods which use only network weights information to determine the redundant filters, are termed as 'Data Independent' methods. Data dependent methods eliminate redundant filters based on the feature map properties like frequency representation [7], rank [8], entropy [10], scale and shift parameters of the batch norm [5], [18] etc. Removal of a filter when the corresponding feature map channel has low class wise mask score for each class [19], prune to minimize feature map reconstruction error [9] these are few examples of existing data dependent filter pruning methods. Most of the data dependent methods [7]–[10] use subsampled training data to estimate the feature map statistics in order to avoid the excessive computational burden of generating feature maps for all training examples. The cost of computing feature map statistics increases proportionally with the spatial size of the input image and also with the number layers and filters used in a CNN model. Subsampled approach works fine when we have taken a significant number of training examples from each class to compute the feature map statistics, dealing with the CIFAR10 dataset which contains 10 classes. However, the performance of these methods severely degrades while dealing with a large diverse dataset like ImageNet which contains 1000 classes as these methods can allow 10images/class to estimate the feature map statistics. Otherwise the computational burden would be very high as the spatial size of the ImageNet data is also large compared to CIFAR10. So data dependent methods suffer from poor estimation of feature map statistics for large diverse dataset like ImageNet, as very low number subsampled data/ class and overall less than 1% of the total training data [7]–[10] is used for estimation. Also additional memory and computations are required for data dependent pruning algorithms like [19] which increases the feature map size by the number of classes or [7] which computes fourier transform of the large feature map matrix.

Data independent methods do not store or process an enormous number of feature maps to compute filter importance as these methods are based on only the learned model weights which remain fixed for all training examples. So the complexity of these methods [12], [14]–[17] are less compared to data dependent methods but the performance of these methods is poor compared to data dependent methods. This happens due to improper analysis of a filter's contribution to generate a feature map. Existing data independent methods [11], [15]–[17], [20] use present layer filter properties only to determine the current layer filter importance which degrades the performance of these methods. However, we find that if we want to minimize the feature map reconstruction error of the next layer in an data independent manner then both the present and the next layer filter information are required to determine the filter importance.

The complexity of the existing data independent or dependent algorithms also increases due to another two reasons- (i) pruning rate per layer is a hyperparameter, (ii) layer by layer prune and retrain. Methods [10], [20], [21] have found out that the different convolutional layers have different sensitivity towards pruning. These methods manually set the pruning rate for each layer after observing the pruning sensitivity of filters in each layer. This requires a lot of manual intervention and the process becomes extremely tedious for deeper models like, ResNet50, ResNet110 etc. So other methods like [11], [15]–[17] prune filters from each layer at a uniform rate in order to avoid tuning of additional hyperparameters but this results in suboptimal performance. This kind of extensive hyperparameter tuning is required for only one reason i.e., the filter importance computed by these methods [15]–[17], [20] are not comparable across the layers. The second problem is layerwise retraining after pruning filters from each layer [9], [22], [23]. This makes the retraining process of the entire network time consuming and also complicated due to tuning of the hyperparameters for retraining each layer.

To deal with all these problems, we propose a novel data independent filter pruning method which prunes filters adaptively from different convolutional layers. Our method calculates the ℓ_1 norm of a filter of the l^{th} layer and also computes the ℓ_1 norm of the corresponding channel of all the filters present in the next layer to compute the importance of the filter. We find that they both contribute to producing feature maps of $(l+1)^{th}$ layer. Our objective in this paper is to design a filter pruning method so that the feature map reconstruction error in the $(l+1)^{th}$ layer is minimum after pruning a filter from l^{th} layer. As we use only the network's kernel (/filter) properties to determine the filter importance, we call our method 'Filter Removal by Analyzing only Network's Kernels (FRANK)'. FRANK does not require layerwise retraining after pruning filters from each layer. Few recent methods [11], [24] have found out that pruning a small fraction of filters from the entire network in multiple epochs is better than the pruning all filters at first and then expecting that the fine tuning of the pruned model with large number of epochs will recover the accuracy. So, our algorithm follows the iterative pruning and retraining procedure until the pruned model matches the given computational budget (FLOPs).

The following are the major contributions of our paper:

1) Contrary to the existing data independent filter pruning methods which prunes filters based on the filter prop-

erties of the current layer only, our in depth analysis of the neural network suggests for the first time in this paper that both the current and the succeeding layer filter information are crucial to determine the filter importance of the current layer. We propose a simple yet effective data independent filter pruning method based on this critical analysis.
2) FRANK adaptively prunes filters from different layers as the filter importance produced by our method are globally comparable across the layers. We follow a simple iterative pruning and retraining process which preserves severe information loss. The hyperparameter values that are used for training the base (/unprune
3) model, we use the same for retraining the pruned model.
4) Although FRANK is neither data driven nor it involves any complex process like time consuming optimization to promote sparsity or use of additional module to determine the pruned network, yet the results on extensive experiments with different datasets (CIFAR, ImageNet) and with different architectures (VGG, ResNet, MobileNet) demonstrates the effectiveness of the FRANK over state-of-the-art (SOTA) filter pruning methods.

II. Related Works

Network parameters can be pruned in two ways - (i) weight pruning, (ii) filter pruning. Some of the existing weight pruning methods are described below.

A. Weight Pruning:

Weight pruning allows to prune weights both from the spatial and the channel dimension of filters. For example, Liu et al. introduced a multi-dimension pruning technique that uses stochastic gradient estimates to simultaneously reduce the number of channels, spatial dimension, and depth of the network [25] whereas Zeng et al. eliminated both interspatial as well as interkernel redundancy after ranking weights using principal component analysis (PCA) [22]. Hu et al. pruned the low valued activation channels after measuring the average proportion of zero activations across all examples and the spatial size [26]. Other weight pruning pruning methods are based on group sparsity for structure regularization [27], 2D Discretre Cosine Transform (DCT) on network weights to minimize the spatial redundancy in a filter [28], second order derivative of the layerwise error to prune parameters from each layer [29].

Although weight pruning removes unimportant weights from the network but it leads to unstructured sparsity. So weight pruning does not provide realistic acceleration without using additional software or hardware. However, filter pruning removes filters from the network and it improves inference speed as the pruned model does not have structural irregularity issues and it can use BLAS libraries directly.

B. Filter Pruning:

Existing filter pruning methods can be divided into mainly four groups based on the type of the pruning algorithm, these are- filter pruning using (i) Filter Attributes, (ii) Feature map Characteristics, (iii) Optimization based methods, (iv) Additional Module or Network.

1) Filter Attributes: Pruning methods that rely on only filter attributes to determine filter importance are also known as data independent methods [12], [14], [16], [17], [20]. Existing methods, such as [20] prunes filters which have a low ℓ_1 filter norm as filters with a low norm generate low activations in the respective CNN feature map. Similarly, He et al. proposed a method that allows a larger optimization space to find the best pruned model by performing soft pruning based on the ℓ_2 norm of the filters of the current layer [17]. He et al. removed those filters which are close to the geometric median of all the filters present in that layer [16], while Singh et al. deleted one of the two strongly correlated filters [14]. Furthermore, Epruner finds optimal pruned architecture by message passing algorithm and using affinity propagation algorithm on the weight matrices [12].

2) Feature Map Characteristics: In contrast to filter attribute based approaches, the pruning methods which use feature map characteristics are termed as data-dependent methods [5], [8], [9], [21], [30]. For example, Ye et al. sparsified the batch normalization layer's scaling value, so more channels were constant for all training samples. They later trimmed these constant channels and adjusted the biases [30]. Luo et al. introduced a data-driven approach for pruning filters from a layer such that the feature map reconstruction error in the next layer is minimized as a result of the pruning [9]. Few recent methods depend on estimating feature rank [8], backpropagating feature selection based relevance score [21], and utilizing activation map distributions [5].

3) Optimization based Methods: Existing methods [4], [6], [31]–[33] which modify the loss function to achieve sparsity but with minimal or no degradation in the performance fall into this category. For instance, Y. Idelbayev. et al. imposed rank constraint on each layer to perform sparse model selection [31]. Likewise, Li et al. modified the loss function and adopted group sparsity regularization [32]. He et al. further explored by alternatively optimizing channel selection and adjusting the retained weights to reduce the feature map reconstruction error locally [33]. In [34], a knee guided evolutionary algorithm is applied to the contradictory objectives of minimizing parameters but maximizing performance. Recently, Tang et al. introduced a regularization method [6] that uses manifold information of training examples for filter removal.

4) Additional Module or Network: Another frequent strategy is to use an extra network or module [23], [35]–[38] to accomplish pruning. Liu et al. introduced an additional meta network to eliminate the least important filters of a base model by using stochastic structure sampling and evolutionary algorithm [38]. In [37], Lin et al. applied generative adversarial learning where the pruned version of the base model served as a generator and an additional fully connected network served as a discriminator. Some recent methods in this area include adding an auxiliary attention layer [36], introducing an episodic memory module and using resampling techniques [35] to extract subnetworks.

FRANK falls into the first category as it is a data independent filter pruning method.

III. Proposed Method

In this section, we will first demonstrate the reason behind using both the current layer and the succeeding layer filter statistics to prune filters from the current layer. To do this, firstly we find out what can be a proper data free pruning metric for a fully connected feed forward neural network (FCFN).

A. Neuron elimination in FCFN

Pruning one filter from l^{th} layer leads to elimination of one feature map from l^{th} layer in case of CNN. Similarly, in FCFN, pruning one filter means deleting a row of $\mathbf{U}(/\mathbf{W}^{[l]})$ matrix and this will also eliminate one neuron (/ feature in CNN) from l^{th} layer. For base (/unpruned) model, $\mathbf{z}^{[l]} = \mathbf{U}\mathbf{a}^{[l-1]}$ where, $\mathbf{U} \in \mathbb{R}^{m\times n}$. For pruned model after elimination of one neuron

$$\widehat{\boldsymbol{z}}^{[l]} = \widehat{\boldsymbol{U}}\mathbf{a}^{[l-1]} \tag{1}$$

where, $\widehat{\boldsymbol{U}} \in \mathbb{R}^{(m-1)\times n}$ and $\widehat{\boldsymbol{z}}^{[l]} \in \mathbb{R}^{m-1}$

Now our objective is to observe the changes that will happen in $\mathbf{V}(/\mathbf{W}^{[l+1]})$ matrix due to one neuron elimination from l^{th} layer. For unpruned model, $\boldsymbol{V} \in \mathbb{R}^{p\times m}$, $\mathbf{z}^{[l+1]} \in \mathbb{R}^{p}$. For pruned model after elimination of one neuron from l^{th} layer, $\widehat{\boldsymbol{V}} \in \mathbb{R}^{p\times(m-1)}$, $\widehat{\boldsymbol{z}}^{[l+1]} \in \mathbb{R}^{p}$.

$$\mathbf{z}^{[l+1]} = \mathbf{V}\mathbf{a}^{[l]} \tag{2}$$

$$\mathbf{z}^{[l+1]} = \begin{bmatrix} V_{11} & V_{12} & \dots & V_{1m} \\ V_{21} & V_{22} & \dots & V_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ V_{p1} & V_{p2} & \dots & V_{pm} \end{bmatrix} \mathbf{a}^{[l]} \tag{3}$$

where, $\mathbf{a}^{[l]} \in \mathbb{R}^{m}$. Elimination of a neuron means one column of the $\mathbf{V}$ matrix has to be deleted as the dimension of $\mathbf{a}^{[l]}$ is reduced by one due to pruning.

Now coming to the pruning criterion based on filter norm, our objective is to find out j^{th} row of $\mathbf{U}$ and j^{th} column of $\mathbf{V}$ such that the feature map in $(l+1)^{th}$ layer i.e., $\widehat{\boldsymbol{z}}^{[l+1]}$ remains close to $\mathbf{z}^{[l+1]}$ even after eliminating j^{th} neuron.

B. Analysis without non-linearity and batch normalization for FCFN

1) Forward propagation: To find out a proper data free pruning criteria based on filter norm, we first analyze the FCFN without batch normalization and non-linearity activation for the sake of simplicity. Here, our objective is to find a pruning metric which can ensure that the feature maps after pruning is close to the unpruned model's feature maps.

If we assume that no non-linearity activation functions and batch norm functions are present in the entire FCFN, then,

$$\mathbf{z}^{[l+1]} = \mathbf{V}\mathbf{z}^{[l]} = \mathbf{V}\mathbf{U}\mathbf{a}^{[l-1]} \tag{4}$$

as $\mathbf{a}^{[l]} = \mathbf{z}^{[l]} = \mathbf{U}\mathbf{a}^{[l-1]}$ if no non-linearity is present. Here, $\mathbf{z}^{[l+1]} \in \mathbb{R}^{p}$, $\mathbf{a}^{[l-1]} \in \mathbb{R}^{n}$

$$\mathbf{z}^{[l+1]} = \begin{bmatrix} | & | & & | \\ \mathbf{V}_{:,1} & \mathbf{V}_{:,2} & \dots & \mathbf{V}_{:,m} \\ | & | & & | \end{bmatrix}^{p\times m} \begin{bmatrix} — \mathbf{U}_{1,:} — \\ — \mathbf{U}_{2,:} — \\ \vdots \\ — \mathbf{U}_{m,:} — \end{bmatrix}^{m\times n} \mathbf{a}^{[l-1]} \tag{5}$$

$$\mathbf{z}^{[l+1]} = \sum_{j=1}^{m} (\mathbf{V}_{:,j}\mathbf{U}_{j,:})\mathbf{a}^{[l-1]} \tag{6}$$

So, if we want to eliminate j_0^{th} neuron from the l^{th} layer, then the feature map at l^{th} layer, then the feature map at $(l+1)^{th}$ layer will be,

$$\widehat{\boldsymbol{z}}^{[l+1]} = \sum_{\substack{j=1 \\ j\neq j_0}}^{m} (\mathbf{V}_{:,j}\mathbf{U}_{j,:})\mathbf{a}^{[l-1]} \tag{7}$$

so,

$$\mathbf{z}^{[l+1]} - \widehat{\boldsymbol{z}}^{[l+1]} = (\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:})\mathbf{a}^{[l-1]} \tag{8}$$

where; $(\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:}) \in \mathbb{R}^{p\times n}$, $\mathbf{V}_{:,j_0} \in \mathbb{R}^{p\times 1}$, $\mathbf{U}_{j_0,:} \in \mathbb{R}^{1\times n}$, $\mathbf{a}^{[l-1]} \in \mathbb{R}^{n\times 1}$.

Ideally if $(\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:})_{k,i} \rightarrow 0\ \forall k,i$ then we should prune j^{th} column of $\mathbf{V}$ (i.e.$\mathbf{V}_{:,j_0}$) and j^{th} row of $\mathbf{U}$ (i.e.$\mathbf{U}_{j_0,:}$). One simple way to check whether all elements of $(\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:})$ is small or not is by taking the absolute sum of all the elements of the matrix $\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:}$. So, we compute the importance corresponding to j_0^{th} neuron ($Imp(j_0)$) using the following equation,

$$\begin{aligned} Imp(j_0) &= \sum_k \sum_i |(\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:})_{k,i}| \\ &= \sum_k \sum_i |(\mathbf{V}_{:,j_0}||\mathbf{U}_{j_0,:}|)_{k,i} \\ &= \|\mathbf{V}_{:,j_0}\|_1 \|\mathbf{U}_{j_0,:}\|_1 \end{aligned} \tag{9}$$

Now we have three possibilities to define the pruning criterion to eliminate a neuron based on the properties of either (a) $\mathbf{U}$ or (b) $\mathbf{V}$ or (c) $\mathbf{UV}$.

Existing methods use ℓ_1 norm of weights of l^{th} layer to eliminate neuron from l^{th} layer. So j_0^{th} neuron is pruned when, $\|\mathbf{U}_{j_0,:}\|_1 < \|\mathbf{U}_{j,:}\|_1\ \forall j \in [1,2,..m];\ j \neq j_0$ However, we have observed through our mathematical analysis that the feature map of $(l+1)^{th}$ layer ($\mathbf{z}^{[l+1]}$) is dependent on both $\mathbf{V}$ and $\mathbf{U}$ matrix. So, another possible way of data free pruning is that prune j_0^{th} neuron if, $\|\mathbf{V}_{:,j_0}\|_1 < \|\mathbf{V}_{:,j}\|_1\ \forall j \in [1,2,..m];\ j \neq j_0$ as $\mathbf{z}^{[l+1]} = \sum_{j=1}^{m} \mathbf{z}_j^{[l]}\mathbf{V}_{:,j}$ However, the problem with the above mentioned pruning criterion is that even if for some $\|\mathbf{V}_{:,j_0}\|_1 < \|\mathbf{V}_{:,j_1}\|_1$ but $\mathbf{z}_{j_0}^{[l]} >> \mathbf{z}_{j_1}^{[l]}$ then the contribution by j_0^{th} neuron to produce $\mathbf{z}^{[l+1]}$ will be greater than the j_1^{th} neuron, even though the norm of the j_0^{th} column is lesser than than the norm of j_1^{th} column of $\mathbf{V}$.

Here we are interested in data free pruning approach, so we do not use the feature map $\mathbf{z}^{[l]}$ to calculate the importance score corresponding to j_0^{th} of l^{th} layer. However, we know that the neurons of l^{th} layer ($\mathbf{z}^{[l]}$) is produced by $\mathbf{U}$ matrix

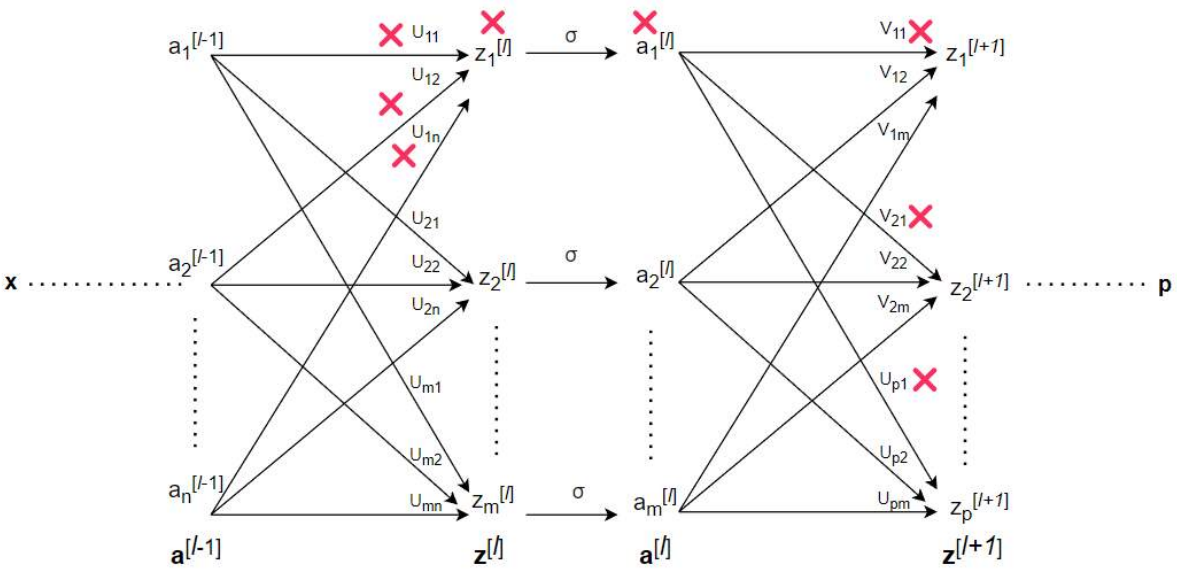

Fig. 2. Neuron elimination from l^{th} layer of a Fully Connected Feedforward Neural network (FCFN). It shows the structural change in $(l+1)^{th}$ layer due to elimination of one neuron from l^{th} layer.

and the neurons (/feature maps) of $(l-1)^{th}$ layer. From eq. 6, we observe that if j_0^{th} row of $\mathbf{U}$ matrix is having low ℓ_1 norm then the j_0^{th} neuron will have less value. A neuron in the l^{th} layer should be pruned, only when both the value of the output of the neuron is small and the contribution to produce the output of the $(l+1)^{th}$ layer is also small. In this analysis of determining importance of a neuron, we do not store or use the output values of neurons rather use only model weights as our method is a data free pruning method. We know that the output of the l^{th} layer is dependent on the model weights of the l^{th} layer. If the model weights to a neuron has low values then the output of the neuron will also have low values. So we use the model weights of the l^{th} layer while determining the importance. Here, we assume that all neurons in $(l-1)^{th}$ layer produce output values in the same scale as it is not observed. We observe that if,

$\|\mathbf{V}_{:,j_0}\|_1\|\mathbf{U}_{j_0,:}\|_1 \approx 0 \implies \sum_k\sum_i |(\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:})_{k,i}| \approx 0 \implies \mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:} \approx \mathbf{0} \implies \mathbf{z}^{[l+1]} - \widehat{\boldsymbol{z}}^{[l+1]} \approx \mathbf{0} \implies \mathbf{z}^{[l+1]} \approx \widehat{\boldsymbol{z}}^{[l+1]}$

So, in practical scenario, our proposed method eliminates j_0^{th} neuron if

$\|\mathbf{V}_{:,j_0}\|_1\|\mathbf{U}_{j_0,:}\|_1 < \|\mathbf{V}_{:,j}\|_1\|\mathbf{U}_{j,:}\|_1 \ \forall j \in [1,2,..m];\ j \neq j_0$

2) Backward propagation: Here we observe the effects of pruning on the gradient of the the model weights while performing back propagation to update the model weights. We observe that,

$$\nabla_{\mathbf{z}^{[l]}}\mathcal{L} = \mathbf{V}^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} \tag{10}$$

$$\nabla_{\mathbf{z}^{[l-1]}}\mathcal{L} = \mathbf{U}^\intercal\nabla_{\mathbf{z}^{[l]}}\mathcal{L} = \mathbf{U}^\intercal\mathbf{V}^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} = (\mathbf{VU})^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} \tag{11}$$

As we know that if we prune j_0^{th} from l^{th} layer using our pruning method, then after pruning,

$$\nabla_{\widehat{\boldsymbol{z}}^{[l-1]}}\mathcal{L} = (\widehat{\boldsymbol{V}}\widehat{\boldsymbol{U}})^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} \approx (\mathbf{VU})^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} \tag{12}$$

as $\mathbf{VU} = \sum_{j=1}^{m}(\mathbf{V}_{:,j}\mathbf{U}_{j,:}) \approx \sum_{\substack{j=1\\ j\neq j_0}}^{m}(\mathbf{V}_{:,j}\mathbf{U}_{j,:}) = \widehat{\boldsymbol{V}}\widehat{\boldsymbol{U}}$

From eq. 11, we observe that the gradient update for all the outputs (similarly, weights) before l^{th} layer will not be hampered significantly even after pruning a neuron from l^{th} layer which satisfies $\|\mathbf{V}_{:,j_0}\|_1\|\mathbf{U}_{j_0,:}\|_1 \approx 0$. In practical scenario, if $\|\mathbf{V}_{:,j_0}\|_1\|\mathbf{U}_{j_0,:}\|_1$ is smaller than $\|\mathbf{V}_{:,j}\|_1\|\mathbf{U}_{j,:}\|_1$ $\forall j \in [1,2,..m];\ j \neq j_0$, then the gradient update for the outputs (similarly, weights) before l^{th} layer will change by smaller amount than pruning a neuron for which importance score is large.

C. Analysis when non-linearity and batch norm is present in FCFN

As the proposed method provides importance score corresponding to a neuron (/feature map). So, even when batch norm and non-linearity is present in the network, we can still compute directly $\|\mathbf{U}_{j_0,:}\|_1$ and $\|\mathbf{V}_{:,j_0}\|_1$ from FCFN.

$\mathbf{a}^{[l-1]} \xrightarrow{\mathbf{U}(/\mathbf{W}^{[l]})} \mathbf{z}^{[l]} \xrightarrow{BN} \mathbf{y}^{[l]} \xrightarrow{ReLU} \mathbf{a}^{[l]} \xrightarrow{\mathbf{V}(/\mathbf{W}^{[l+1]})} \mathbf{z}^{[l+1]}$

If, $\|\mathbf{U}_{j_0,:}\|_1 \approx 0 \implies \mathbf{z}_{j_0}^{[l]} \approx 0(\forall$ training examples)

$\mathbf{z}_{j_0}^{[l]} \approx 0 \implies \mathbf{y}_{j_0}^{[l]} \approx 0 \implies \mathbf{a}_{j_0}^{[l]} \approx 0(\forall$ training examples)

Now along with $\|\mathbf{U}_{j_0,:}\|_1 \approx 0$ if, $\|\mathbf{V}_{:,j_0}\|_1 \approx 0$, then both $\mathbf{a}_{j_0}^{[l]} \approx \mathbf{0}$ and $\mathbf{V}_{:,j_0} \approx \mathbf{0}$

$$\mathbf{z}^{[l+1]} = (\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:})\mathbf{a}^{[l-1]} + \sum_{\substack{j=1 \\ j\neq j_0}}^{m} (\mathbf{V}_{:,j}\mathbf{U}_{j,:})\mathbf{a}^{[l-1]} \tag{13}$$

With the proposed pruning criteria, both $\mathbf{V}_{:,j_0} \approx \mathbf{0}$ and $\mathbf{a}^{[l]}_{j_0} \approx \mathbf{0}$, which indicates that $\mathbf{V}_{:,j_0}\mathbf{a}^{[l]}_{j_0} \approx \mathbf{0}$. So, even after pruning j_0^{th} neuron from l^{th} layer, the output vector ($\widehat{\boldsymbol{z}}^{[l+1]}$) remains approximately same as the output vector ($\mathbf{z}^{[l+1]}$) produced by the the base model for $(l+1)^{th}$ layer.

D. Feature map elimination in CNN using FRANK

Here,

$$\begin{aligned} \mathbf{f}^{[l]}_j &\in \mathbb{R}^{n\times s\times s} \\ \mathbf{f}^{[l]} &\in \mathbb{R}^{m\times n\times s\times s} \\ \mathbf{f}^{[l+1]}_k &\in \mathbb{R}^{m\times s\times s} \\ \mathbf{f}^{[l+1]} &\in \mathbb{R}^{p\times m\times s\times s} \end{aligned} \tag{14}$$

Likewise, for a single training example, the size of the feature map becomes
$\mathbf{a}^{[l-1]} \in \mathbb{R}^{n\times w\times h}, \mathbf{a}^{[l]} \in \mathbb{R}^{m\times w\times h}, \mathbf{a}^{[l+1]} \in \mathbb{R}^{p\times w\times h}$

w, h indicates the width and height of a feature map and s indicates the kernel size of a filter. $w, hands$ can vary over the layers. However, for notational simplicity, we do not use w_l, h_l and s_l for l^{th} layer and w_{l+1}, h_{l+1} and s_{l+1} for $(l+1)^{th}$ layer. Our filter pruning method does not modify the spatial size of features or filters, so the spatial size of the features or filters remain same even after pruning. Only the number of input and output channels gets modified in each layer due to pruning.

$$\mathbf{z}^{[l+1]} = \begin{bmatrix} \mathbf{z}^{[l+1]}_1 \\ \mathbf{z}^{[l+1]}_2 \\ \vdots \\ \mathbf{z}^{[l+1]}_p \end{bmatrix} \tag{15}$$

where, $\mathbf{z}^{[l+1]} \in \mathbb{R}^{p\times w\times h}$ and $\mathbf{z}^{[l+1]}_k \in \mathbb{R}^{w\times h}$

In CNN, l^{th} layer activation map ($\mathbf{a}^{[l]}$) is convolved with the k^{th} filter of $(l+1)^{th}$ layer and it produces k^{th} feature map ($\mathbf{z}^{[l+1]}_k$) of $(l+1)^{th}$ layer,

$$\mathbf{z}^{[l+1]}_k = \mathbf{f}^{[l+1]}_k * \mathbf{a}^{[l]} = \sum_{j=1}^{m} \mathbf{f}^{[l+1]}_{k_j} * \mathbf{a}^{[l]}_j \tag{16}$$

where, $\mathbf{f}^{[l+1]}_{k_j} \in \mathbb{R}^{s\times s}, \mathbf{a}^{[l]}_j \in \mathbb{R}^{w\times h}$

Each channel of a filter is convolved with the corresponding channel of the feature map and the output for each channel is summed to produce a feature map as shown in eq. 16. Now if we prune j_0^{th} filter from the l^{th} layer then j_0^{th} feature map of l^{th} layer will also be eliminated. However, the size of the feature map for $(l+1)^{th}$ layer will not be affected.

Here, k^{th} feature map of $(l+1)^{th}$ layer is denoted by $\mathbf{z}^{[l+1]}_k$ for base (/unpruned) model and by $\widehat{\boldsymbol{z}}^{[l+1]}_k$ for pruned model when j_0^{th} filter is pruned from l^{th} layer.

$$\widehat{\boldsymbol{z}}^{[l+1]}_k = \sum_{\substack{j=1 \\ j\neq j_0}}^{m} \mathbf{f}^{[l+1]}_{k_j} * \mathbf{a}^{[l]}_j \tag{17}$$

So,

$$\mathbf{z}^{[l+1]}_k - \widehat{\boldsymbol{z}}^{[l+1]}_k = \mathbf{f}^{[l+1]}_{k_{j_0}} * \mathbf{a}^{[l]}_{j_0} \tag{18}$$

We want to prune j_0^{th} filter from l^{th} layer in such a manner that, $\mathbf{z}^{[l+1]}_k - \widehat{\boldsymbol{z}}^{[l+1]}_k \approx \mathbf{0}\ \forall k \in [1, 2, .., p]$. This would only be possible when $\mathbf{f}^{[l+1]}_{k_{j_0}} \approx \mathbf{0}$ and $\mathbf{a}^{[l]}_{j_0} \approx \mathbf{0}$ (any tensor $\mathbf{T} \approx \mathbf{0} \implies$ all coefficients of $\mathbf{T} \approx 0$). As we are interested in data free filter pruning method, so we do not use $\mathbf{a}^{[l]}_j$ to decide the index of the filter which needs to pruned.

$$\mathbf{z}^{[l]}_{j_0} = \mathbf{f}^{[l]}_{j_0} * \mathbf{a}^{[l-1]} \tag{19}$$

We know that $\|\mathbf{f}_{j_0}\|^{(*)}_1 \approx 0 \implies \mathbf{z}^{[l]}_{j_0} \approx 0 \implies \mathbf{a}^{[l]}_{j_0} \approx 0(\forall$ training examples) where, $\mathbf{f}^{[l]}_{j_0} \in \mathbb{R}^{n\times s\times s}, \mathbf{z}^{[l]}_{j_0} \in \mathbb{R}^{w\times h}$

$(*)$ Here, in this paper, ℓ_1 norm of any matrix (/tensor) is computed by first flattening (/vectorizing) the matrix (/tensor) into 1D vector and thereafter taking the ℓ_1 norm of that vector. For example, if a tensor $\mathbf{T} \in \mathbb{R}^{a\times b\times c}$, then $\|\mathbf{T}\|_1 = \sum_i \sum_j \sum_k |\mathbf{T}_{ijk}|$. We have used this notation only for convenience. It is not an operator norm of a matrix (/tensor).

So, in practical scenario, our proposed method prunes j_0^{th} filter from l^{th} layer, when $\|\mathbf{f}^{[l]}_{j_0,:}\|_1 \|\mathbf{f}^{[l+1]}_{:,j_0}\|_1 < \|\mathbf{f}^{[l]}_{j,:}\|_1 \|\mathbf{f}^{[l+1]}_{:,j}\|_1 \forall j \in [1, 2, .., m]$ where, $\mathbf{f}^{[l]}_{j_0,:} \in \mathbb{R}^{1\times n\times s\times s}, \mathbf{f}^{[l+1]}_{:,j_0} \in \mathbb{R}^{p\times 1\times s\times s}$ We can observe using our pruning criteria that if j_0^{th} filter is having small norm and also the corresponding input channel of every filter in the next layer is having small norm then the contribution of the j_0^{th} filter in l^{th} layer to produce $\mathbf{z}^{[l+1]}$ will be lower than all the other filters present in the l^{th} layer. Our method first computes the multiplication of $\|\mathbf{f}^{[l]}_{j_0,:}\|_1$ and $\|\mathbf{f}^{[l+1]}_{:,j_0}\|_1$ for j_0^{th} filter of l^{th} layer to compute the importance score of that filter. we present a block diagram of the FRANK pipeline in Fig. 3.

E. Globally comparable normalized filter importance score

In convolutional neural networks, different convolutional layers have different number of filters. In initial layers, low level features that are common to almost all classes are extracted using few convolutional filters and then more abstract features are generated from these features using convolutional filters of deeper layers. We generally observe that the number of filters in a convolutional layer increases as the layer becomes deeper in order to produce more discriminative features. For example, in VGG16, the first convolutional layer has 64 filters, whereas the tenth convolutional layer contains 512 filters. It indicates that 64 filters in the first layer are responsible for producing a feature map of the second layer, while 512 filters in the tenth layer contribute to producing a feature map of the eleventh layer. So to calculate the effective

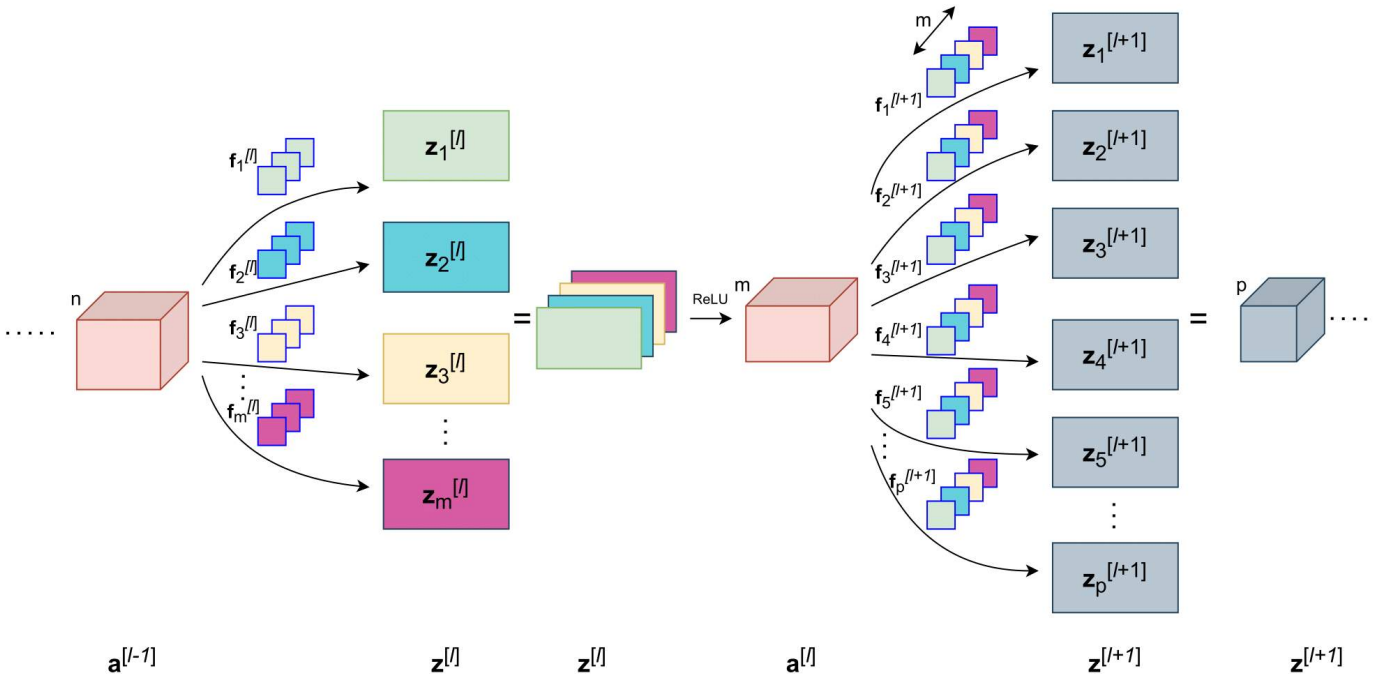

Fig. 3. Pipeline of FRANK. (i) Low $\|\mathbf{f}_{1,:}^{[l]}\|_1$ (green, i.e. first filter of l^{th} layer) indicates low $\|\mathbf{z}_1^{[l]}\|_1$, (ii) Now if $\|\mathbf{f}_{:,1}^{[l+1]}\|_1$ (green, i.e. first channel of all the filters of $l+1^{th}$ layer) is also low, then $\mathbf{f}_{1,:}^{[l]}$ should be pruned as both the first feature $\mathbf{z}_1^{[l]}$ and $\mathbf{f}_{:,1}^{[l+1]}$ has not contributed significantly to produce feature map of $(l+1)^{th}$ layer. However, if one of the ℓ_1 norm is large then the contribution of the first filter of l^{th} layer need not be necessarily low as both the l^{th} (present) layer and $(l+1)^{th}$ (next) layer kernel (/filter) contribute to generate $\mathbf{z}^{[l+1]}$. So, FRANK takes both $\|\mathbf{f}_{j,:}^{[l]}\|_1$ and $\|\mathbf{f}_{:,j}^{[l+1]}\|_1$ to determine the j^{th} filter that needs to be pruned from l^{th} layer. Here, eliminating $\mathbf{f}_{1,:}^{[l]}$ imposes elimination of $\mathbf{f}_{:,1}^{[l+1]}$ to match the structural size after pruning.

contribution of a filter, FRANK normalizes the importance score by the number of filters present in a layer so that their contribution can be compared across the layers. Proposed method computes the normalized importance score ($I_{j_0}^{[l]}$) of the j_0^{th} filter of the l^{th} layer by the following equation,

$$I_{j_0}^{[l]} = \frac{\|\mathbf{f}_{j_0,:}^{[l]}\|_1 \|\mathbf{f}_{:,j_0}^{[l+1]}\|_1}{m} \tag{20}$$

where, m indicates the number of filters present in l^{th} layer. Once the importance score of each filter has been computed with the Eq. 20, then we prune the least important filters from the network. We summarize the proposed method in Algorithm 1

IV. Experiments

In this section, we show the observation results obtained from the experiments on filter pruning. Further, we illustrate the performance of GFI based adaptive pruning (GFI-AP) method.

A. Experimental Settings

1) Dataset and Architecture: We conduct extensive experiments on three benchmark datasets, i.e., CIFAR10 [ ], CIFAR100 [ ] and ILSVRC-2012 [ ] while using different types of convolutional networks as a base model which includes single branch network (VGG), multi branch network (ResNet) and compact network (MobileNet). Here, two different datasets CIFAR10 and CIFAR100 both have a total of 60,000 images (50k training, 10k testing) but CIFAR10 contains 10 classes and 6000 images/class whereas CIFAR100 contains 100 classes and 600 images/class. ILSVRC-2012 is a large scale diverse dataset having 1000 classes, containing 1.28 million training images and 50k validation images.

2) Training Configuration: We use Stochastic Gradient Descent (SGD) optimizer with the batch size of 256 to train all the base models. For experiments with CIFAR datasets, we use a 300 epochs training schedule with an initial learning rate of 0.1 for the first 150 epochs and then divided by 10 at 150th and 225th epoch. The momentum and weight decay are set to 0.9 and 5e-4 respectively for CIFAR experiments. We follow the same training schedule with the same hyperparameters for both VGG16 and Residual Nets (56,110) while dealing with CIFAR datasets. Similarly, for experiments with ILSVRC-2012, we set the momentum to 0.875 and weight decay to 1/32768 for MobileNetV2 and we use Pytorch's pretrained model for ResNet50. Cosine scheduler of 150 epochs with an initial learning rate of 0.05 and final learning rate of 1e-6 is used for training MobileNetV2. We use the two common data augmentation techniques i.e, random crop and horizontal flip which is the same as Pytorch's official example for ILSVRC-2012 experiments.

3) Pruning and Retraining Configuration: We prune a small but fixed percentage of filters in every training epoch

Algorithm 1 Algorithm Description of FRANK

Input: CNN model (**M**); Desired FLOPs reduction (D%)
Given: Prune filters per epoch (F%)
Output: Compressed model $\widehat{M^*}$ after pruning and retraining

for all epoch e = 1 to E **do**
  if $e == 1$ **then**
    Initialize current FLOPs reduction percentage $C = 0$
    Target FLOPs achieved flag $T = 0$
    Train the model or load pretrained state M^*
  else
    Load M^*_{e-1} as the base model
  end if
  if $T == 0$ **then**
    Compute filter importance for each filter with Eq.20
    Sort all filters as per their importance score
    Obtain M_e after trimming F% least important filters
    Compute C=100*(1-$\frac{\text{CountFLOPs}(M_e)}{\text{CountFLOPs}(M)}$)
    if $C <= D$ **then**
      $T = 0$
    else
      $T = 1$
    end if
  else
    Do not prune further
  end if
  Fine tune the small model for one epoch and obtain M^*_e
end for

until the desired FLOPs condition is satisfied. We do not perform extensive tuning of hyperparameters to retrain or finetune the pruned model, rather we simply follow the same training schedule with the same hyperparameters as training of the base model to perform iterative prune and retrain. Once the model meets the desired FLOPs condition, the rest of the training epochs is used only to finetune the final pruned model. For CIFAR experiments, we prune filters in such a manner that 1% of filters from the entire network are pruned in every epoch. However, we follow a smaller number of training epochs for the ILSVRC-12 dataset, i.e., 100 epochs for ResNet50 and 150 epochs for MobileNetV2 compared to 300 epochs while using CIFAR datasets. So we prune more filters in every epoch while dealing with the ILSVRC-12 dataset, specifically 2.5% filters from the model. This ensures that the final pruned model gets sufficient epochs for fine tuning. FRANK can also be applied to the base model when it is not pretrained. In these cases, we start pruning once the model is trained for a few number of epochs, specifically 5% of the total epochs in all experiments.

B. Results and Analysis

Here we compare the performance of FRANK with several state of the art (SOTA) pruning methods including LRMF [7], MFP [11], Filter Sketch [13], LFPC [39], HRank [8], ASFP [15], SFP [17], FPGM [16], GAL [37]. Although FRANK is a simple data-independent filter pruning method but this comparison includes all types of recent filter pruning methods irrespective of whether it is data dependent or optimization based methods. We demonstrate the superiority of FRANK over existing methods in terms of performance in this section.

TABLE I
PRUNING RESULTS OF RESNET56 ON CIFAR10.

Method	Unpruned Top1 Acc (%)	Pruned Top1 Acc (%)	Acc. drop (%)	FLOPs rdcn (%)
CP [33]	92.8	91.8	1.00	50
DCP [40]	93.8	93.49	0.31	50
HRank [8]	93.26	93.17	0.09	50
ASFP [15]	93.59	93.12	0.47	52.6
SFP [17]	93.59	93.35	0.24	52.6
MFP [11]	93.59	93.56	0.03	52.6
LRMF [7]	93.59	93.25	0.34	52.6
FPGM [16]	93.59	93.26	0.33	52.6
LFPC [39]	93.59	93.24	0.35	52.9
FRANK	**93.57**	**93.42**	**0.15**	**53.4**

TABLE II
PRUNING RESULTS OF RESNET110 ON CIFAR10.

Method	Unpruned Top1 Acc (%)	Pruned Top1 Acc (%)	Acc. drop (%)	FLOPs rdcn (%)
PFEC [20]	93.53	93.3	0.23	38.6
GAL [37]	93.5	92.74	0.76	48.5
SFP [17]	93.68	92.90	0.78	52.3
ASFP [15]	93.68	93.1	0.58	52.3
MFP [11]	93.68	93.31	0.37	52.3
FPGM [16]	93.68	93.74	-0.06	52.3
LRMF [7]	93.68	93.88	-0.20	52.3
HRank [8]	93.5	93.36	0.14	58.2
LFPC [39]	93.68	93.07	0.61	60.3
FRANK	**93.81**	**93.74**	**0.07**	**60.8**

1) ResNet on CIFAR10: We apply FRANK on multi-branch networks like ResNet56, ResNet110 while using CIFAR10 dataset and observe the pruning results in Table I and Table II respectively. FRANK reduces the computational burden of ResNet56 by more than half (53.4%) with a small reduction of 0.15% in top 1 accuracy as shown in Table I. Similarly, Table II shows that FRANK reduces 60.8% FLOPs for ResNet110_CIFAR10 configuration but still provides top-1 accuracy of 93.74% which is better than the recent methods like LRMF [7], LFPC [39], HRank [8] and many others. Moreover, FRANK performs way better than the existing filter norm based methods like PFEC [20], SFP [17], ASFP [15], for example- SFP decreases 52.3% FLOPs with the accuracy drop of 0.78% whereas FRANK provides 60.8% FLOPs reduction with a very small amount of 0.07% accuracy drop.

2) VGG on CIFAR datasets: We also apply our method on single branch networks like VGG16 which have thirteen

TABLE III
PRUNING RESULTS OF VGG16 ON CIFAR DATASETS.

Config	Method	Unpruned Top1 Acc (%)	Pruned Top1 Acc (%)	Acc. drop (%)	FLOPs rdcn (%)
VGG16 CIFAR10	PFEC [20]	93.25	93.4	-0.15	34.2
	GAL [37]	93.96	93.42	0.54	45.2
	HRank [8]	93.96	93.43	0.53	53.5
	FRANK	**93.9**	**93.97**	**-0.07**	**53.6**
VGG16 CIFAR100	GFI-AP [41]	73.97	73.68	0.29	53.5
	FRANK	**74.29**	**74.3**	**-0.01**	**53.6**

TABLE IV
PRUNING RESULTS OF MOBILENETV2 ON IMAGENET.

Method	Unpruned Top1 Acc (%)	Pruned Top1 Acc (%)	Acc. drop (%)	FLOPs rdcn (%)
FRANK	**71.82**	**71.43**	**0.39**	**20.5**
DCP [40]	70.11	64.22	5.89	44.8
FRANK	**71.82**	**68.92**	**2.9**	**44.8**

convolutional layers and three fully connected layers. Table III indicates that FRANK improves over baseline accuracy by 0.07% for CIFAR10 while lowering computational burden by 53.6%. Furthermore, even for 100 class classification with the CIFAR100 dataset, FRANK does not degrade the model performance of VGG16 while saving more than 50% FLOPs. Under the similar FLOPs constraint, FRANK outperforms the existing methods like HRank [8] which decreases the classification accuracy by 0.53% whereas FRANK increases the classification accuracy by 0.07% for CIFAR10.

3) ResNet50 on ImageNet: Now, we evaluate the performance of FRANK when pruning ResNet50, which is utilized for the ILSVRC-2012 dataset. It is difficult to prune ResNet50 on ILSVRC-2012 since it is a large-scale dataset having 1000 classes. However, ResNet-50 is often used to evaluate pruning algorithms. Here we observe from Table V that FRANK reduces 42.7% FLOPs but yields improvement in top-1 accuracy by 0.14% over baseline for ResNet50_ILSVRC-2012 configuration. Table V also shows that FRANK achieves 55% FLOPs reduction while decreasing the top-1 accuracy by 0.69% whereas existing methods like RRBP [ ] decrease the top-1 by 3.15%, FPGM by 2.02% and MFP by 1.29% even with lesser than 55% FLOPs reduction. As seen in Fig. 1, FRANK consistently outperforms SOTA and delivers superior top-1 accuracy for varying FLOPs reduction rates. This highlights the superiority of FRANK over other pruning algorithms even for challenging datasets like ImageNet.

4) MobileNetV2 on ImageNet: MobileNetV2 is one of the most efficient networks especially designed for mobile and edge devices that uses low computational power. The use of depth-wise separable convolution and inverted residual blocks makes mobileNetV2 a compact architecture. So, pruning MobileNetV2 is extremely challenging, especially when applied to the ILSVRC-2012 dataset.. However, FRANK can still reduce more than 20% FLOPs with only 0.39% top-1 accuracy drop for MobileNetv2_ImageNet configuration as shown in Table IV. Moreover, it provides a 3% improvement in terms of top-1 accuracy drop over competing methods like DCP when the same percentage of FLOPs reduction is performed (44.8%).

C. More Explorations

In this section, we first apply FRANK on models which are not pretrained. We start with a base model when it is randomly initialized. Here, we use 5% of the total training epochs for training the base model, and we start iterative pruning and retraining from the next epoch. As our starting base model is at random state, we call this method FRANK-NP, indicating FRANK applied on a model which is not pretrained.

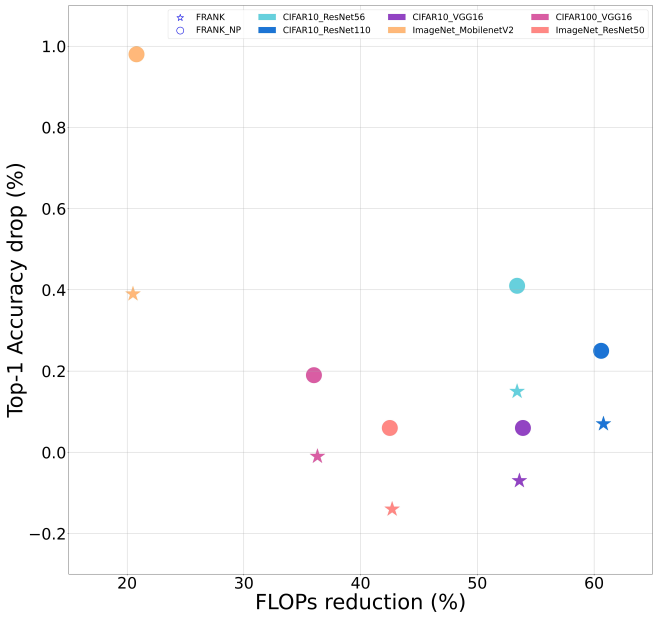

Fig. 4. Performance comparison between FRANK and FRANK_NP for different datasets and architectures. FRANK_NP means when FRANK applied to a base model which is Not Pretrained. Top-1 accuracy drop for both FRANK and FRANK-NP are very close to each other for similar amount of FLOPs reduction for all configurations ecxcept ImageNet_MobileNet configuration.

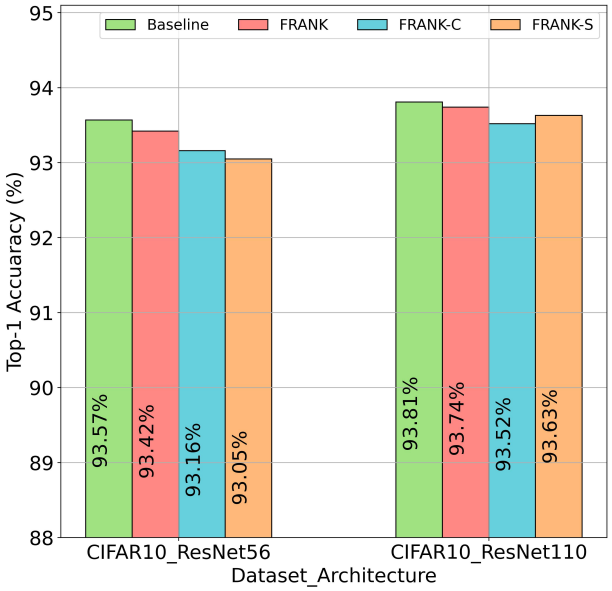

Fig. 5. Performance comparison between FRANK, FRANK_C and FRANK_S for ResNet56 and ResNet110 for CIFAR10 classification. FRANK_C indicates when only the current layer filter information is used for filter importance. similarly, FRANK_S indicates when only the succeeding layer filter information is used for filter importance. FRANK outperforms both FRANK-C and FRANK-S in all cases as it uses both the current and the succeeding layer filter information to compute filter importance.

TABLE V
FLOPS COMPARISON ACROSS DIFFERENT PRUNING METHODS FOR IMAGENET CLASSIFICATION WITH RESNET50.

Method	Unpruned FLOPs (B)	Unpruned Top-1 Acc.	Pruned Top-1 Acc.	Top-1 Acc. ↓	Unpruned Top-5 Acc.	Pruned Top-5 Acc.	Top-5 Acc. ↓	Pruned FLOPs (B)	FLOPs ↓ (%)
SSS-26 [42]	4.09	76.15%	74.18%	1.97%	92.96%	91.91%	1.05%	2.82	31.90%
CP [33]	4.09	76.15%	72.30%	3.85%	92.96%	90.80%	2.16%	2.73	34.10%
FilterSketch- 0.7 [13]	4.09	76.13%	75.22%	0.91%	92.86%	92.41%	0.45%	2.64	35.50%
SFP [17]	4.09	76.15%	74.61%	1.54%	92.87%	92.06%	0.81%	2.39	41.80%
Epruner-0.71 [12]	4.13	76.01%	74.95%	1.06%	92.96%	92.36%	0.60%	2.37	42.60%
MFP 30% [11]	4.09	76.15%	75.67%	0.48%	92.87%	92.81%	0.06%	2.36	42.20%
FRANK 0.42	**4.12**	**76.15%**	**76.29%**	**-0.14%**	**92.87%**	**92.93%**	**-0.06%**	**2.36**	**42.70%**
GAL [37]	4.09	76.15%	71.95%	4.20%	92.96%	90.79%	2.17%	2.33	43.70%
SSS-32 [42]	4.09	76.12%	71.82%	4.30%	92.86%	90.79%	2.07%	2.33	43.70%
HRank [8]	4.09	76.15%	75.01%	1.14%	92.96%	92.33%	0.63%	2.3	43.90%
FilterSketch- 0.6 [13]	4.09	76.13%	74.68%	1.45%	92.86%	92.17%	0.69%	2.23	45.50%
White-Box [19]	4.09	76.15%	75.32%	0.83%	92.96%	92.43%	0.53%	2.22	45.60%
ABP [36]	3.89	75.88%	74.80%	1.08%	92.76%	92.36%	0.40%	2.21	42.80%
GFI-AP [41]	3.89	75.95%	74.61%	1.34%	92.89%	92.01%	0.88%	2.01	48.33%
MetaPruning [38]	4.09	76.60%	75.40%	1.20%				2	48.70%
FPGM [16]	4.09	76.15%	74.13%	2.02%	92.96%	92.87%	0.09%	1.9	53.50%
MFP 40% [11]	4.09	76.15%	74.86%	1.29%	92.87%	92.43%	0.44%	1.9	53.50%
RRBP [43]	4.09	76.15%	73.00%	3.15%	92.96%	91.00%	1.96%	1.86	54.50%
FRANK 0.55	**4.12**	**76.15%**	**75.46%**	**0.69%**	**92.87%**	**92.66%**	**0.21%**	**1.85**	**55.10%**
GAL [37]	4.09	76.15%	71.80%	4.35%	92.96%	90.82%	2.14%	1.84	55.60%
ThiNet 50 [9]	3.86	75.30%	72.03%	3.27%	92.20%	90.99%	1.21%	1.71	55.80%
LFPC [39]	4.09	76.15%	74.18%	1.97%	92.96%	91.92%	1.04%	1.61	60.80%
HRank [8]	4.09	76.15%	71.98%	4.17%	92.96%	91.01%	1.95%	1.55	62.60%
FilterSketch- 0.4 [13]	4.09	76.13%	73.04%	3.09%	92.86%	91.18%	1.68%	1.51	63.10%
White-Box [19]	4.09	76.15%	74.21%	1.94%	92.96%	92.01%	0.95%	1.5	63.50%
FRANK 0.63	**4.12**	**76.15%**	**74.79%**	**1.36%**	**92.87%**	**92.26%**	**0.62%**	**1.49**	**63.90%**
Epruner-0.81 [12]	4.13	76.01%	72.73%	3.28%	92.96%	91.01%	1.95%	1.29	68.80%
FRANK 0.69	**4.12**	**76.15%**	**74.08%**	**2.07%**	**92.87%**	**91.93%**	**0.94%**	**1.25**	**69.70%**

We observe from Fig. 4 that the Top-1 accuracy drop for FRANK and FRANK-NP is very close to each other under similar FLOPs constraint for most of the dataset_architecture configuration; for example- the top-1 accuracy drop compared to baseline is 0.07% for FRANK whereas it is 0.25% for FRANK-NP while reducing more than 60% FLOPs in both cases for CIFAR10_ResNet110 configuration. Similarly, FRANK provides a -0.14% top-1 accuracy drop and FRANK-NP achieves a 0.06% top-1 accuracy drop while reducing 43% FLOPs for ResNet50_ImageNet configuration. However, the difference between FRANK and FRANK-NP becomes 0.59% with only 20% FLOPs reduction while utilizing MobileNetV2 for ImageNet classification. It indicates that the impact of pretraining becomes prominent only when both the base model is relatively small and the dataset is challenging. So, FRANK can efficiently prune the base models even if they are not pretrained.

Now, we investigate the consequences of including filter contributions from both the current and the succeeding layer, as opposed to either the current or succeeding layer. In Fig. 111, FRANK-C is used when the contribution of a filter belonging to the current layer is only used to determine the filter importance whereas FRANK-S indicates when the channel contribution of filters from the succeeding layer is only used to determine the filter importance of the current layer. Fig. 5 shows that FRANK which uses both the current and the succeeding layer filter contribution outperforms FRANK-C and FRANK-S for both CIFAR10_ResNet56 and CIFAR10_ResNet110 configuration. FRANK provides 0.26% improvement over FRANK-C and 0.37% improvement over FRANK-S while pruning 53% FLOPs for CIFAR10_ResNet56 configuration. Similarly, when we reduce 61% FLOPs from ResNet110 while performing CIFAR10 classification, we observe that FRANK-S provides better results than FRANK-C. However, FRANK outperforms FRANK-C and FRANK-S also in this case.

V. CONCLUSION

The proposed method (FRANK) selects the filters for pruning while assuring minimum feature map reconstruction error for each layer. However, the most attractive element of our technique is that we do not use any feature map information to achieve this. We find that a filter which belongs to the present layer and its corresponding channel in all the filters belong to the next layer both are directly responsible for generating a feature map of the next layer. Hence, our method considers both contributions when estimating the importance of a filter, as opposed to previous methods that rely only on the current layer information. We also perform iterative pruning and retraining so that a large number of filters are not pruned at a single epoch which results in an unrecoverable accuracy drop. However, the superiority of FRANK over existing methods is that it does not require layerwise retraining, rigorous hyperparameter search for fine-tuning, or human intervention to set the pruning percentage for each layer. Rather, it automatically prunes filters until the target FLOPs budget is fulfilled. Although FRANK is a simple data-free pruning method but it outperforms the SOTA filter pruning

methods. For instance- FRANK lowers 42.7% FLOPs without sacrificing accuracy, whereas the best competing method decreases top-1 accuracy by 0.48% under the similar FLOPs constraint for ResNet50_ImageNet configuration.

REFERENCES

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

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] 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.

[4] J. Shi, J. Xu, K. Tasaka, and Z. Chen, "Sasl: Saliency-adaptive sparsity learning for neural network acceleration," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 5, pp. 2008–2019, 2020.

[5] M. Kang and B. Han, "Operation-aware soft channel pruning using differentiable masks," in *International Conference on Machine Learning*, PMLR, 2020, pp. 5122–5131.

[6] Y. Tang, Y. Wang, Y. Xu, *et al.*, "Manifold regularized dynamic network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5018–5028.

[7] X. Zhang, W. Xie, Y. Li, J. Lei, and Q. Du, "Filter pruning via learned representation median in the frequency domain," *IEEE Transactions on Cybernetics*, 2021.

[8] M. Lin, R. Ji, Y. Wang, *et al.*, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1529–1538.

[9] 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*, vol. 41, no. 10, pp. 2525–2538, 2018.

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

[11] Y. He, P. Liu, L. Zhu, and Y. Yang, "Filter pruning by switching to neighboring cnns with good attributes," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[12] M. Lin, R. Ji, S. Li, *et al.*, "Network pruning using adaptive exemplar filters," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[13] M. Lin, L. Cao, S. Li, *et al.*, "Filter sketch for network pruning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[14] P. Singh, V. K. Verma, P. Rai, and V. Namboodiri, "Leveraging filter correlations for deep model compression," in *Proceedings of the IEEE/CVF Winter Conference on applications of computer vision*, 2020, pp. 835–844.

[15] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang, "Asymptotic soft filter pruning for deep convolutional neural networks," *IEEE transactions on cybernetics*, 2019.

[16] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4340–4349.

[17] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866*, 2018.

[18] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[19] Y. Zhang, M. Lin, C.-W. Lin, *et al.*, "Carrying out cnn channel pruning in a white box," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

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

[21] R. Yu, A. Li, C.-F. Chen, *et al.*, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

[22] L. Zeng and X. Tian, "Accelerating convolutional neural networks by removing interspatial and interkernel redundancies," *IEEE transactions on cybernetics*, vol. 50, no. 2, pp. 452–464, 2018.

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

[24] J. Guo, W. Zhang, W. Ouyang, and D. Xu, "Model compression using progressive channel pruning," *IEEE Transactions on Circuits and Systems for Video Technology*, 2020.

[25] Z. Liu, X. Zhang, Z. Shen, Y. Wei, K.-T. Cheng, and J. Sun, "Joint multi-dimension pruning via numerical gradient update," *IEEE Transactions on Image Processing*, vol. 30, pp. 8034–8045, 2021.

[26] 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.

[27] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[28] Z. Liu, J. Xu, X. Peng, and R. Xiong, "Frequency-domain dynamic pruning for convolutional neural net-

works," in *Advances in Neural Information Processing Systems*, 2018, pp. 1043–1053.

[29] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *Advances in Neural Information Processing Systems*, vol. 30, pp. 4857–4867, 2017.

[30] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," *arXiv preprint arXiv:1802.00124*, 2018.

[31] Y. Idelbayev and M. A. Carreira-Perpinán, "Low-rank compression of neural nets: Learning the rank of each layer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8049–8059.

[32] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 8018–8027.

[33] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1389–1397.

[34] Y. Zhou, G. G. Yen, and Z. Yi, "A knee-guided evolutionary algorithm for compressing deep neural networks," *IEEE transactions on cybernetics*, vol. 51, no. 3, pp. 1626–1638, 2019.

[35] S. Gao, F. Huang, W. Cai, and H. Huang, "Network pruning via performance maximization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9270–9280.

[36] G. Tian, Y. Sun, Y. Liu, *et al.*, "Adding before pruning: Sparse filter fusion for deep convolutional neural networks via auxiliary attention," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[37] S. Lin, R. Ji, C. Yan, *et al.*, "Towards optimal structured cnn pruning via generative adversarial learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2790–2799.

[38] Z. Liu, H. Mu, X. Zhang, *et al.*, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3296–3305.

[39] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2009–2018.

[40] Z. Zhuang, M. Tan, B. Zhuang, *et al.*, "Discrimination-aware channel pruning for deep neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 875–886.

[41] M. Mondal, B. Das, S. D. Roy, P. Singh, B. Lall, and S. D. Joshi, "Adaptive cnn filter pruning using global importance metric," *Computer Vision and Image Understanding*, vol. 222, p. 103 511, 2022.

[42] 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.

[43] Y. Zhou, Y. Zhang, Y. Wang, and Q. Tian, "Accelerate cnn via recursive bayesian pruning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3306–3315.