Computer Vision and Image Understanding
journal homepage: www.elsevier.com

Feature Independent Filter Pruning by Successive Layers Analysis

Milton Mondal[a,1,**], Bishshoy Das[a,1], Brejesh Lall[a], Pushpendra Singh[b], Sumantra Dutta Roy[a], Shiv Dutt Joshi[a]

[a]*Department of Electrical Engineering, Indian Institute of Technology Delhi, Hauz Khas, New Delhi - 110016, India*
[b]*Department of Electronics and Communication Engineering, National Institute of Technology Hamirpur, Hamirpur (HP) - 177005, India*

ABSTRACT

Convolutional neural networks (CNNs) have become deeper and wider over time. However due to low computational power, mobile devices or embedded systems cannot use very deep models. Filter pruning solves this by eliminating redundant filters. Pruning can be performed in a feature dependent or independent manner. Feature dependent methods require extensive time to determine the filter importance as these methods require generation and processing of feature maps for each example. Additionally, in iterative pruning, filter importance is computed several times based on the current state. This increases algorithm execution time further. However, existing feature independent methods are fast, but they perform poor as they compute importance using only current layer filter weights. However, our analysis suggests that both the current and succeeding layer filters are crucial to determine filter importance. We propose 'Filter Pruning by Successive Layers analysis' (FPSL), a novel feature independent algorithm, that considers the effect of pruning a filter on the generation of feature maps for the first time. Moreover, FPSL does not require layer-wise retraining, rigorous hyperparameter search for fine-tuning, or human intervention to set the pruning percentage per layer. These make FPSL extremely fast, efficient, and adaptive. Thus it follows iterative pruning and retraining. FPSL outperforms the state-of-the-art (SOTA) methods on extensive experiments with different datasets (CIFAR, ImageNet) and architectures (VGG, ResNet, MobileNet). It decreases the computational burden of VGG16 by half but improves CIFAR10 and CIFAR100 accuracy. Even for ImageNet, FPSL reduces 42.7% floating point operations (FLOPs) while maintaining top-1 accuracy for ResNet50.

1. Introduction

Convolutional neural networks (CNNs) have been extremely successful in solving diverse computer vision tasks, which include image classification [45], object detection [51], image captioning [25], 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. Workstations and GPU servers are able to use CNNs 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 [49, 32], (b) tensor decomposition [6, 21], (c) knowledge distillation [14, 1], (d) compact network synthesis [2, 38], (e) pruning network parameters [35, 55], etc. Out of all these solutions, pruning network parameters reduces the computational cost of deep neural networks (DNNs) by first identifying and then deleting 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 flexibil-

**Corresponding Author
e-mail: milton.mondal@ee.iitd.ac.in (Milton Mondal), bishshoy.das@ee.iitd.ac.in (Bishshoy Das), brejesh@ee.iitd.ac.in (Brejesh Lall), spushp@nith.ac.in (Pushpendra Singh), sumantra@ee.iitd.ac.in (Sumantra Dutta Roy), sdjoshi@ee.iitd.ac.in (Shiv Dutt Joshi)
[1]Equal Contribution

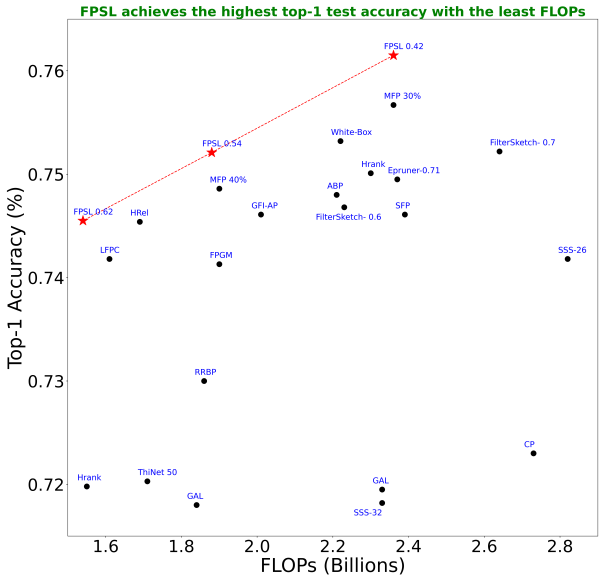

Fig. 1. Top-1 accuracy vs FLOPs for different pruning methods for ResNet50 while using ImageNet dataset. FPSL provides the best test accuracy compared to state of the art (SOTA) methods for similar target FLOPs.

ity to the user to choose a standard base model (like VGG [45], ResNet [7], MobileNet [42], etc.), and thereafter it automatically generates a compact model from the base model by trimming the redundant parameters.

Researchers have proposed mainly three kinds of pruning methods, (i) weight pruning, (ii) filter pruning and (iii) intermediate level pruning. Weight pruning methods delete weights of low importance 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. In intermediate level pruning, a group of filter channels (block sparsity) or a group of filter weights (N:M sparsity) within a filter channel are removed. However, the tracking and updating process of group of weights during the block level pruning makes the entire process complex, whereas specialized sparse tensor core based GPU is required to obtain N:M sparsity. 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.

A filter pruning method can either be (i) Feature Dependent [44, 20, 47, 56, 28, 37, 36], or (ii) Feature Independent [12, 27, 26, 46, 9, 11, 10]. The methods that use feature map information to find the filter importance are referred to as 'Feature Dependent' methods, whereas the methods that use only filter attributes and no feature map properties to determine the redundant filters are referred to as 'Feature Independent' methods. In both cases low importance filters are pruned and then retraining is followed to recover the performance. Feature dependent methods eliminate redundant filters based on the feature map properties like frequency representation [56], rank [28], entropy [36], scale and shift parameters of the batch norm [31, 20] etc. Removal of a filter when the corresponding feature map channel has low class-wise mask score for each class [57], prune to minimize feature map reconstruction error [37]. These are few examples of existing feature dependent filter pruning methods. Most of the feature dependent methods [56, 28, 37, 36] use subsampled training data to estimate the feature map statistics in order to avoid the excessive time and computational burden of generating and processing feature maps for all training examples at any instant. 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. Subsampled approach works fine when we have taken a significant number of training examples from each class to compute the feature map statistics, like dealing with the CIFAR10 [22] dataset which contains 10 classes. However the performance of these methods severely degrades while dealing with a large and diverse datasets like ImageNet [41] which contains 1000 classes, as these methods can allow only 10 images per class to estimate the feature map statistics. Otherwise the time required to process those feature maps to estimate properties like rank or class-wise activation would be very high as the spatial size of the ImageNet data is also large compared to CIFAR10. So feature dependent methods suffer from poor estimation of feature map statistics for large diverse datasets like ImageNet, as very few samples per class is used for estimation which is overall less than 1% of the total training data [56, 28, 37, 36]. Also additional memory and computations is required for feature dependent pruning algorithms in which feature map sizes increase by the number of classes [57] or which computes fourier transforms of large feature map matrices [56]. Whereas, feature independent methods do not store or process an enormous number of feature maps to compute filter importance scores as these methods are based only on the model's weights which remains fixed for all training examples. So the complexity of these methods [27, 46, 9, 11, 10] are less compared to feature dependent methods but the performance of these methods are often poor compared to feature dependent methods. This happens due to improper analysis of a filter's contribution towards generating feature maps. Existing feature independent methods [12, 9, 11, 10, 23] use only present layer filter properties to determine the current layer filter importances which degrade the performance of these methods as these methods do not consider the impact of pruning a filter on feature map. We find that removing a filter from a layer not only reduces the number of filters in that layer by one but it also enforces reduction in the number of input channels of all filters in the succeeding layer by one. After pruning, the number of features in the current layer reduces by one but the number of features remain same in the succeeding layer as shown in Fig. 2. The size of the filters and features of all other layers remain unaffected. Our analysis indicates that both the current and the immediate next layer's filter information are required to determine the filter that needs to be

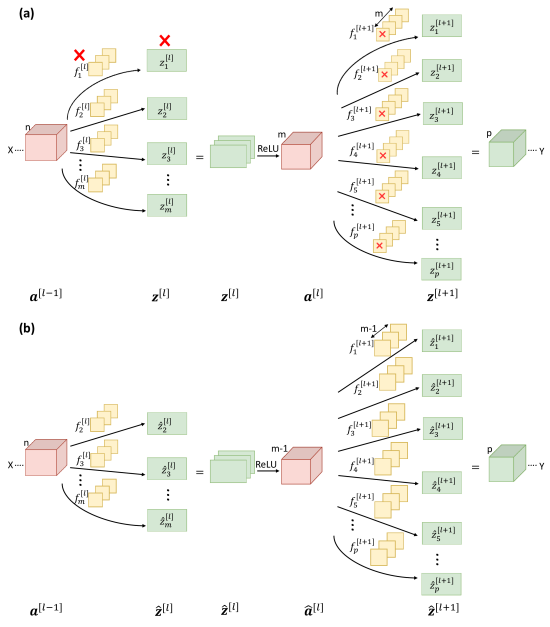

Fig. 2. Pruning a single filter from a CNN. (a) unpruned model (b) pruned model. Eliminating the first filter of l^{th} layer, not only removes the first feature map of l^{th} layer but also reduces channel dimension of all the filters in $(l+1)^{th}$ layer by one. However the number of features remain same in $(l+1)^{th}$ layer even after pruning.

prune from current layer. The pruned model recover its performance during the retraining phase as we prevent the model to deviate significantly from its previously converged optima.

The complexity of the existing feature independent or dependent algorithms increase due to two additional reasons - (i) using pruning rate per layer as a hyperparameter, (ii) pruning and retraining on a per layer basis. Methods [54, 23, 36] 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. Such extensive hyperparameter tuning is required because filter importance metrics computed by these methods [23, 9, 11, 10] are not comparable across the layers. The second problem is layer-wise retraining after pruning [55, 37, 16]. This makes the retraining process of the entire network time consuming and also complicated due to the the additional overhead of hyperparameter search for each layer. So other methods such as [12, 9, 11, 10] prune filters from each layer at a uniform rate in order to avoid tuning of additional hyperparameters but this results in suboptimal performance. Few recent methods [12, 5] 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. For feature dependent methods like [56, 28, 37, 36], iterative pruning and retraining becomes extremely challenging and time taking as the filter importance has to be computed several times based on the current state of the model.

To address the aforementioned problems, we propose a novel feature independent filter pruning method which prunes filters adaptively from different convolutional layers. Pruning a filter from a layer leads to elimination of one feature map of that layer and the channel depth of all the filters in the immediate layer reduces by one. However, the number of feature maps in the immediate next layer remains the same even after pruning. So while pruning a filter from a layer, our objective is that even after pruning, the feature maps in the immediate next layer remain close to unpruned state feature maps. During retraining of the pruned model, the likelihood of regaining performance improves with lesser change in the feature map due to pruning. So, 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 immediate next layer to compute the importance of filters. We find that they both contribute to producing feature maps of $(l+1)^{th}$ layer. We aim to design a filter pruning method such that the feature maps of $(l+1)^{th}$ layer is minimally affected while pruning a filter from l^{th} layer. As we use only the filter properties of two successive layers to determine a filter's importance, we name our method as 'Filter Pruning by Successive Layers analysis' (FPSL). The attractive element of FPSL is that it is feature independent, so we do not have to pass all the input images to the network to determine the filter importance at any time. We follow the iterative pruning and retraining procedure until the pruned model matches a target computational budget (FLOPs). The following are the major contributions of our method:

1. We propose a novel filter pruning method 'FPSL' based on our analysis which suggests that both current and the immediate next layer are crucial for determining the importance of any filter of the current layer.
2. FPSL computes filter importance extremely fast as it requires only the filter weights of two successive layers. Additionally, it does not require layer-wise retraining, or human intervention to set the pruning percentage on a per-layer basis. It adaptively prunes filters from different convolutional layers.
3. Extensive experiments with various datasets (CIFAR, ImageNet) and architectures (VGG, ResNet, MobileNet) demonstrate that FPSL provides better classification performance compared to existing filter pruning methods for similar FLOPs reduction.

Empirical results also demonstrate the effectiveness of FPSL across various network architectures. For VGG16, FPSL provides 54% FLOPs reduction without any drop in test accuracy both for CIFAR10 and CIFAR100 classification tasks. Similarly, FPSL can maintain the same performance as the unpruned network and can still reduce 50% FLOPs from both ResNet56 and ResNet110 for CIFAR10 classification. Even for the more challenging ResNet50_ImageNet configuration, FPSL reduces 43% FLOPs without any drop in the test accuracy.

2. Related Works

Network parameters can be pruned in three ways - (i) weight pruning, (ii) intermediate level pruning and (iii) filter pruning.

2.1. Weight Pruning:

Weight pruning allows pruning of weights both from spatial and channel dimensions of filters. For example, Liu et al. [35] introduce a multi-dimension pruning technique that uses stochastic gradient estimates to simultaneously reduce the number of channels, spatial dimension, and depth of the network whereas Zeng et al. [55] eliminate both interspatial as well as interkernel redundancy after ranking weights using principal component analysis (PCA). Hu et al. [15] prune low valued activation channels after measuring the average proportion of zero activations across all examples and the spatial size. Other weight pruning pruning methods are based on group sparsity for structure regularization [50], 2D Discretre Cosine Transform (DCT) on network weights to minimize the spatial redundancy in a filter [34], second order derivative of the layer-wise error to prune parameters from each layer [3].

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.

2.2. Intermediate Level Pruning:

Recently, few pruning methods are proposed which perform intermediate level pruning [60, 58, 59, 19, 29]. These pruning methods either remove a block or a group of weights from the network. In block-sparsity based methods [19, 29], block removal means elimination of all the weights within a specific filter channel. For instance, Yu Ji et al.[19] prune a particular channel of consecutive filters and follows 1*N pattern of pruning, while M. Lin et al. [29] reorder the filter channels and then removes the smaller magnitude filter channels for structured pruning. Whereas N:M sparsity based methods impose a structured sparsity pattern to prune a group of weights like N number of weights from a group of M weights within a filter channel.

Recent methods solved few dominant complexities in the implementation of N:M sparsity for example reducing the training cost by approximating the gradients of removed weights using straight through estimator (STE) [60] or by learning the best weight combination [58], by overcoming the dense gradient computation in the backward path by disentangaling the forward and backward paths [59]. Generally N:M fine grained structured sparsity provides both computational efficiency and lossless performance. However, N:M sparsity based acceleration is possible only when we use NVIDIA A100 GPU as it supports sparse Matrix Multiply Accumulate (MMA) instructions [29]. All other available GPUs except NVIDIA A100 do not have the sparse tensor cores. So we are not able to use N:M sparsity based approaches on any other platforms.

2.3. 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.

2.3.1. Filter Attributes

Pruning methods that rely on only filter attributes to determine filter importance are also known as feature independent methods [27, 46, 11, 10, 23]. Existing methods such as PFEC [23] 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. [10] propose 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. In FPGM, He et al.[11] remove filters which are close to the geometric median of all the filters present in a given layer, while Singh et al. [46] delete one of two strongly correlated filters. Furthermore, in Epruner [27], the optimal pruned architecture is found by a message passing algorithm and using affinity propagation algorithm on weight matrices.

2.3.2. Feature Map Characteristics

In contrast to filter attribute based approaches, pruning methods which use feature map characteristics are termed as feature-dependent methods [20, 28, 37, 53, 54]. For example, Ye et al. [53] sparsify the batch normalization layer's scaling value, so more channels become constant for all training samples. These constant channels are later trimmed and biases are adjusted. Luo et al. [37] introduce 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. Other methods include dependencies on feature rank estimtes [28], backpropagating feature selection based relevance scores [54], and utilization of activation map distributions [20].

2.3.3. Optimization based Methods

Existing methods [44, 47, 18, 24, 13] which modify the loss function to achieve sparsity but with minimal or no degradation in the performance fall in this category. For instance, Y. Idelbayev. et al. [18] impose rank constraint on each layer to perform sparse model selection. Likewise Li et al.[24] modify the loss function and adopted group sparsity regularization. He et al.[13] further explore alternatively optimizing channel selection and adjusting the retained weights to reduce the feature map reconstruction error locally. In [61], a knee guided evolutionary algorithm is applied to the contradictory objectives of minimizing parameters but maximizing performance. Tang et al. [47] introduce a regularization method that uses manifold information of training examples for filter removal.

2.3.4. Using Additional Modules or Networks

Another frequent strategy is to use an extra network or module [4, 48, 30, 33, 16] to accomplish pruning. Adding an extra network or module for pruning relates to multiple knowledge representation [52]. The original network contains the overall knowledge and functionality, while the specialized pruning network or module contains pruning-specific knowledge. Combining these representations helps in optimizing the pruning process. Liu et al.[33] introduce an additional meta network

to eliminate the least important filters of a base model by using stochastic structure sampling and evolutionary algorithm. Lin et al.[30] apply generative adversarial learning where the pruned version of the base model serves as a generator and an additional fully connected network serves as a discriminator. Some recent methods in this area include adding an auxiliary attention layer [48], introducing an episodic memory module and using resampling techniques [4] to extract subnetworks.

FPSL falls into the first category as it is a filter attributes based filter pruning method. It stands out as a distinctive feature independent method that specifically accounts for the consequences of pruning a filter from a layer on the generation of feature maps of the immediate next layer. This sets FPSL apart from other feature independent methods, as it recognizes the interconnectedness between filters across consecutive layers while taking pruning decisions. Unlike existing filter attributes based methods that consider only present layer filter properties [12, 9, 11, 10, 23], FPSL takes into account both the current layer and the immediate next layer's filter information to determine the filter that needs to be pruned. Our method also eliminates the need for layer-wise retraining after pruning filters from each layer, thereby avoiding the additional overhead of hyperparameter search for each layer. This advantage makes FPSL stand out from existing methods [55, 37, 16]. Furthermore, FPSL provides the benefit of generating globally comparable importance scores for filters across all layers. This feature allows FPSL to automatically and adaptively determine the pruning fraction specific to each layer without manual intervention, distinguishing it from existing approaches [23, 9, 11, 10]. All these advantages make FPSL a convenient and efficient method for producing a compact CNN.

3. Proposed Method

In this section, we first demonstrate the rationale behind using both the current layer's and the succeeding layer's filter information to prune filters from the current layer and thereafter we describe the proposed method. Firstly we find out what can be a proper feature independent pruning metric for a fully connected feed forward neural network (FCFN).

3.1. Neuron elimination in FCFN

In FCFN, the base model generates the neurons ($\mathbf{z}^{[l]}$) of l^{th} layer by the following equation,

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

$$\mathbf{z}^{[l]} = \begin{bmatrix} U_{11} & U_{12} & \dots & U_{1n} \\ U_{21} & U_{22} & \dots & U_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ U_{m1} & U_{m2} & \dots & U_{mn} \end{bmatrix} \mathbf{a}^{[l-1]} \tag{2}$$

where, $\mathbf{z}^{[l]} \in \mathbb{R}^m$, $\mathbf{a}^{[l-1]} \in \mathbb{R}^n$, $\mathbf{U} \in \mathbb{R}^{m \times n}$

Similarly, $(l+1)^{th}$ layer neurons ($\mathbf{z}^{[l+1]}$) are generated by the following equation,

$$\mathbf{z}^{[l+1]} = \mathbf{V}\mathbf{a}^{[l]} \tag{3}$$

$$\mathbf{z}^{[l+1]} = \begin{bmatrix} V_{11} & V_{12} & \dots & V_{1m} \\ V_{21} & V_{22} & \dots & V_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ V_{p1} & V_{p2} & \dots & V_{pm} \end{bmatrix} \mathbf{a}^{[l]} \tag{4}$$

where, $\mathbf{z}^{[l+1]} \in \mathbb{R}^p$, $\mathbf{a}^{[l]} \in \mathbb{R}^m$, $\mathbf{V} \in \mathbb{R}^{p \times m}$

In CNN, pruning one filter from the l^{th} layer leads to the elimination of one feature map from the l^{th} layer. Similarly, deleting a row of $\mathbf{U}(\mathbf{W}^{[l]})$ matrix in FCFN is analogous to the process of pruning a filter in CNN. Deleting a row of $\mathbf{U}$ will also eliminate one neuron (feature in CNN) from l^{th} layer as shown in Fig. 3. The feature maps of the pruned model after elimination of one neuron are described by,

$$\widehat{\mathbf{z}}^{[l]} = \widehat{\boldsymbol{U}}\mathbf{a}^{[l-1]} \tag{5}$$

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

Elimination of a neuron enforces that one column of the $\mathbf{V}$ matrix needs to be deleted as the dimension of $\mathbf{a}^{[l]}$ gets reduced by one due to pruning. So the size of the weight matrix after the elimination of one neuron from l^{th} layer would be, $\widehat{\boldsymbol{V}} \in \mathbb{R}^{p\times(m-1)}$. However the size of both $\widehat{\mathbf{z}}^{[l+1]}$ and $\mathbf{z}^{[l+1]}$ would be same, i.e. $\widehat{\mathbf{z}}^{[l+1]} \in \mathbb{R}^p$.

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

3.2.1. Forward propagation

To find out a proper feature independent 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,

$$\mathbf{z}^{[l+1]} = \mathbf{V}\mathbf{z}^{[l]} = \mathbf{V}\mathbf{U}\mathbf{a}^{[l-1]} \tag{6}$$

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

$$\mathbf{z}^{[l+1]} = \begin{bmatrix} | & | & & | \\ \mathbf{V}_{:,1} & \mathbf{V}_{:,2} & \dots & \mathbf{V}_{:,m} \\ | & | & & | \end{bmatrix}^{p\times m} \begin{bmatrix} — & \mathbf{U}_{1,:} & — \\ — & \mathbf{U}_{2,:} & — \\ & \vdots & \\ — & \mathbf{U}_{m,:} & — \end{bmatrix}^{m\times n} \mathbf{a}^{[l-1]} \tag{7}$$

Here, we express the matrix multiplication $\mathbf{VU}$ as the summation of matrices obtained by multiplying columns of V by the corresponding rows of U.

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

Fig. 3 shows how the elimination of the first neuron from the l^{th} layer leads to the removal of all the incoming connections from $(l-1)^{th}$ layer to that neuron (i.e., $\mathbf{U}_{1,:}$) and also removal of all the outgoing connections from that neuron to $(l+1)^{th}$ layer (i.e., $\mathbf{V}_{:,1}$). Similarly, if we want to eliminate j_0^{th} neuron from the l^{th}

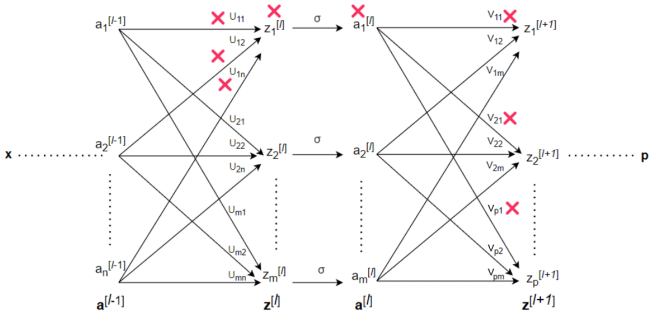

Fig. 3. Neuron elimination from l^{th} layer of a Fully Connected Feedforward Neural network (FCFN). It shows the structural change in $(l+1)^{th}$ layer due to elimination of one neuron from l^{th} layer.

layer, both j_0^{th} column of $\mathbf{V}$ ($\mathbf{V}_{:,j_0}$) and j_0^{th} row of $\mathbf{U}$ ($\mathbf{U}_{j_0,:}$) will be pruned and the feature maps at $(l+1)^{th}$ layer after pruning j_0^{th} neuron will be,

$$\widehat{\mathbf{z}}^{[l+1]} = (\sum_{\substack{j=1 \\ j\neq j_0}}^{m} \mathbf{V}_{:,j}\mathbf{U}_{j,:})\mathbf{a}^{[l-1]} \tag{9}$$

so

$$\mathbf{z}^{[l+1]} - \widehat{\mathbf{z}}^{[l+1]} = (\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:})\mathbf{a}^{[l-1]} \tag{10}$$

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

Here, our objective is to find out that j_0^{th} index for which elements of $(\mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:})$ are small. So, we observe the ℓ_1 norm of both $\mathbf{V}_{:,j_0}$ and $\mathbf{U}_{j_0,:}$ and compute the importance of j_0^{th} index ($Imp(j_0)$) using the following equation,

$$Imp(j_0) = \|\mathbf{V}_{:,j_0}\|_1\|\mathbf{U}_{j_0,:}\|_1 \tag{11}$$

One can eliminate a neuron based on the properties of either (a) $\mathbf{U}$ or (b) $\mathbf{V}$ or (c) $\mathbf{UV}$. j_0^{th} neuron can be pruned when, $\|\mathbf{U}_{j_0,:}\|_1 < \|\mathbf{U}_{j,:}\|_1\ \forall j \in [1,2,..m];\ j \neq j_0$. Existing methods [9, 10, 23] use this approach which is based on only $\mathbf{U}_{j_0,:}$ to eliminate j_0^{th} neuron from l^{th} layer. However if $\|\mathbf{V}_{:,j_0}\|_1 >> \|\mathbf{V}_{:,j}\|_1$ then the contribution by j_0^{th} neuron to produce $\mathbf{z}^{[l+1]}$ will be greater than the j^{th} neuron, even though $\|\mathbf{U}_{j_0,:}\|_1 < \|\mathbf{U}_{j,:}\|_1$. So our method considers both $\mathbf{U}_{j_0,:}$ and $\mathbf{V}_{:,j_0}$ to determine the j_0^{th} neuron that needs to be pruned. We find that if, $\|\mathbf{V}_{:,j_0}\|_1\|\mathbf{U}_{j_0,:}\|_1 \approx 0 \implies \mathbf{V}_{:,j_0}\mathbf{U}_{j_0,:} \approx \mathbf{0} \implies \mathbf{z}^{[l+1]} \approx \widehat{\mathbf{z}}^{[l+1]}$

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

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

3.2.2. Backward propagation

Here we observe the effects of pruning on the gradient of the model weights while performing backpropagation to update the weights. We observe that,

$$\nabla_{\mathbf{z}^{[l]}}\mathcal{L} = \mathbf{V}^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} \tag{12}$$

$$\nabla_{\mathbf{z}^{[l-1]}}\mathcal{L} = \mathbf{U}^\intercal\nabla_{\mathbf{z}^{[l]}}\mathcal{L} = \mathbf{U}^\intercal\mathbf{V}^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} = (\mathbf{VU})^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} \tag{13}$$

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

$$\nabla_{\widehat{\mathbf{z}}^{[l-1]}}\mathcal{L} = (\widehat{\mathbf{V}}\widehat{\mathbf{U}})^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} \approx (\mathbf{VU})^\intercal\nabla_{\mathbf{z}^{[l+1]}}\mathcal{L} \tag{14}$$

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

From eq. 13, 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 and weights before l^{th} layer will change by smaller amount than pruning a neuron for which importance score is large. So the pruned network can recover the performance with little retraining as it has not deviated from its optima significantly.

3.3. FCFN with non-linearity and batch norm

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 $\mathbf{V}_{:,j_0} \approx \mathbf{0}$.

Proposed method performs pruning of j_0^{th} neuron when both $\mathbf{a}_{j_0}^{[l]} \approx \mathbf{0}$ and $\mathbf{V}_{:,j_0} \approx \mathbf{0}$ as it uses both $\|\mathbf{U}_{j_0,:}\|_1$ and $\|\mathbf{V}_{:,j_0}\|_1$. So, even after pruning j_0^{th} neuron from l^{th} layer, the output vector ($\widehat{\mathbf{z}}^{[l+1]}$) remains approximately same as the output vector ($\mathbf{z}^{[l+1]}$) produced by the the base model for $(l+1)^{th}$ layer. We find that the proposed importance score is easy to compute and no modification is required even when non-linearity and batch norm layer is present in the network.

3.4. Feature map elimination in CNN using FPSL

We extend our analysis to determine the filter pruning criterion for CNN. Here, the l^{th} layer activation maps ($\mathbf{a}^{[l]}$) is convolved with the k^{th} filter of $(l+1)^{th}$ layer and it produces k^{th} feature map ($\mathbf{z}_k^{[l+1]}$) of $(l+1)^{th}$ layer,

$$\mathbf{z}_k^{[l+1]} = \mathbf{f}_k^{[l+1]} * \mathbf{a}^{[l]} = \sum_{j=1}^{m}\mathbf{f}_{k_j}^{[l+1]} * \mathbf{a}_j^{[l]} \tag{15}$$

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

In this paper, $*$ symbol is used to represent the convolution operator. Here, w, h indicate the width and height of a feature map and $s\times s$ indicates the kernel size of a filter and each of them 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. FPSL does not modify the spatial size of features or filters. So the spatial size remains same, only

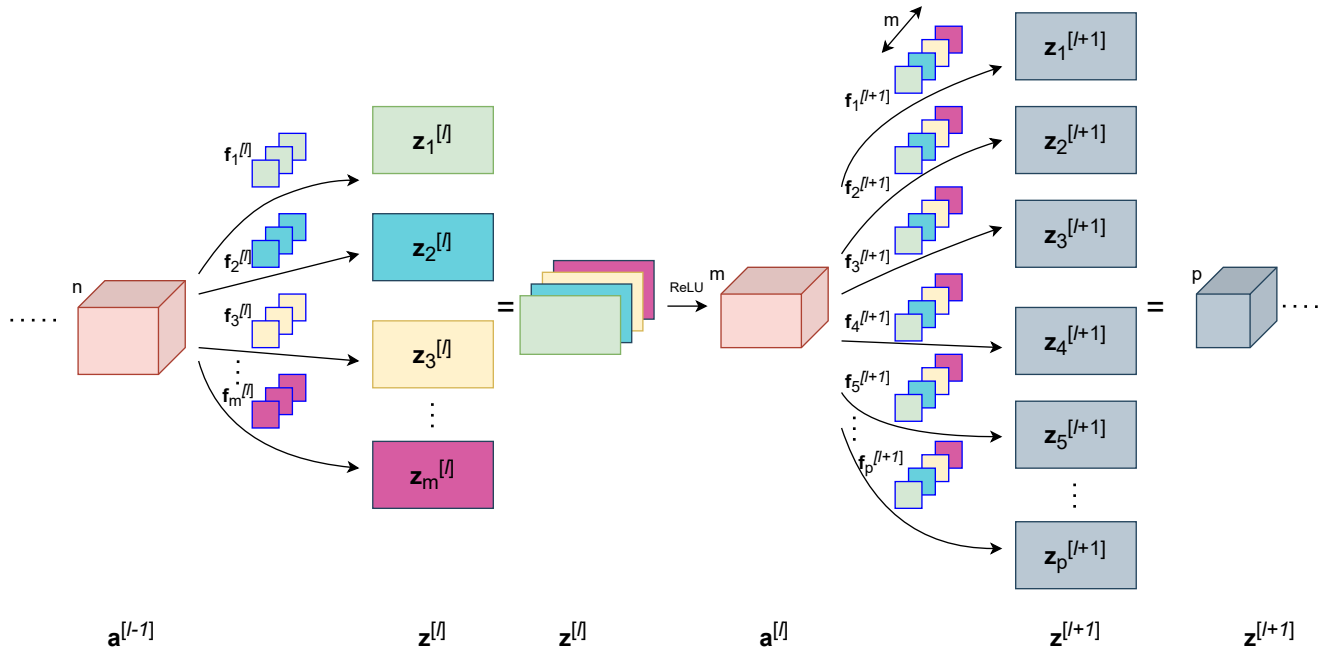

Fig. 4. Pipeline of FPSL. (i) Low $\|\mathbf{f}_{1,:}^{[l]}\|_1$ (green, i.e. first filter of l^{th} layer) indicates low $\|\mathbf{z}_1^{[l]}\|_1$, (ii) If $\|\mathbf{f}_{:,1}^{[l+1]}\|_1$ (green, i.e. first channel of all the filters of $(l+1)^{th}$ layer) is also low, then $\mathbf{f}_{1,:}^{[l]}$ 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 maps 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 filter contribute to generate $\mathbf{z}^{[l+1]}$. So FPSL 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. In filter pruning, eliminating $\mathbf{f}_{1,:}^{[l]}$ imposes elimination of $\mathbf{f}_{:,1}^{[l+1]}$ to match the structural size after pruning. Here, $\mathbf{f}_j^{[l]} = \mathbf{f}_{j,:}^{[l]} \in \mathbb{R}^{1\times n\times s\times s}$ and $\mathbf{f}_{:,j}^{[l+1]} \in \mathbb{R}^{p\times 1\times s\times s}$.

the number of input and output channels gets modified due to pruning.

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. 15. 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. So after pruning j_0^{th} filter,

$$\widehat{\mathbf{z}}_k^{[l+1]} = \sum_{\substack{j=1 \\ j\neq j_0}}^{m} \mathbf{f}_{k_j}^{[l+1]} * \mathbf{a}_j^{[l]} \tag{16}$$

$$\mathbf{z}_k^{[l+1]} - \widehat{\mathbf{z}}_k^{[l+1]} = \mathbf{f}_{k_{j_0}}^{[l+1]} * \mathbf{a}_{j_0}^{[l]} \tag{17}$$

Here, k^{th} feature map of $(l+1)^{th}$ layer is denoted by $\mathbf{z}_k^{[l+1]}$ for base model and by $\widehat{\mathbf{z}}_k^{[l+1]}$ for pruned model when j_0^{th} filter is pruned from l^{th} layer. Our objective is to prune j_0^{th} filter from l^{th} layer in such a manner that, $\mathbf{z}_k^{[l+1]} - \widehat{\mathbf{z}}_k^{[l+1]} \approx \mathbf{0}\ \forall k \in [1,2,..,p]$. This would only be possible when $\mathbf{f}_{k_{j_0}}^{[l+1]} \approx \mathbf{0}$ and $\mathbf{a}_{j_0}^{[l]} \approx \mathbf{0}$ (any tensor $\mathbf{T} \approx \mathbf{0} \implies$ all coefficients of $\mathbf{T} \approx 0$). Here, we are interested in feature independent filter pruning method, so we use $\|\mathbf{f}_{j_0}^{[l]}\|_1$ instead of $\mathbf{a}_{j_0}^{[l]}$ to decide the index of the filter which needs to pruned.

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

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

So in practical scenario, our proposed method prunes j_0^{th} filter from l^{th} layer, when $\|\mathbf{f}_{j_0,:}^{[l]}\|_1 \|\mathbf{f}_{:,j_0}^{[l+1]}\|_1 < \|\mathbf{f}_{j,:}^{[l]}\|_1 \|\mathbf{f}_{:,j}^{[l+1]}\|_1 \forall\, j \in [1,2,..,m]$ where, $\mathbf{f}_{j_0,:}^{[l]} \in \mathbb{R}^{1\times n\times s\times s}, \mathbf{f}_{:,j_0}^{[l+1]} \in \mathbb{R}^{p\times 1\times s\times s}$. 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. So FPSL first computes the multiplication of $\|\mathbf{f}_{j_0,:}^{[l]}\|_1$ and $\|\mathbf{f}_{:,j_0}^{[l+1]}\|_1$ to compute the importance score of j_0^{th} filter of l^{th} layer. We also present a block diagram of the FPSL pipeline in Fig. 4 to provide a visual representation for better understanding.

(#) In this paper, ℓ_1 norm of any matrix (tensor) is computed by first flattening (vectorizing) the matrix 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.

3.5. 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. In order to calculate the effective contribution of a filter, FPSL 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,

$$I_{j_0}^{[l]} = \frac{||\mathbf{f}_{j_0,:}^{[l]}||_1 ||\mathbf{f}_{:,j_0}^{[l+1]}||_1}{m} \tag{19}$$

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. 19, then we prune the least important filters from the network. We summarize the proposed method in Algorithm 1

Algorithm 1: Algorithm Description of FPSL

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

```
for 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 T == 0 then
        Compute filter importance for each filter with Eq. 19
        Sort all filters as per their importance score
        Obtain M_e after trimming F% least important filters
        Compute C = 100*(1- CountFLOPs(M_e) / CountFLOPs(M))
        if C <= D then
            T = 0
        else
            T = 1
        end
    else
        Do not prune further
    end
    Fine tune the small model for one epoch and obtain M*_e
end
```

4. Experiments

In this section, we show the observation results obtained from the experiments on filter pruning. Further, we illustrate the performance of FPSL for different datasets and architectures.

4.1. Experimental Setup

4.1.1. Training Configuration

We use Stochastic Gradient Descent (SGD) optimizer with a batch size of 256 to train all base models. For experiments with CIFAR datasets, we employ a 300-epoch training schedule with an initial learning rate of 0.1 for the first 150 epochs and then divided it by 10 at the 150^{th} and at the 225^{th} epoch. VGG16 and Residual Nets (ResNet56, ResNet110) are trained using the same hyperparameters for CIFAR experiments. The momentum is set to 0.9 for all our experiments. The weight decay is set at 5×10^{-4} and 1×10^{-4} for CIFAR and ImageNet experiments respectively. For ImageNet dataset, FPSL directly loads PyTorch's [40] pretrained model for ResNet50 like [4] and MobileNetV2 uses a 150 epochs training schedule with an initial learning rate of 0.04 and then divided by 2 at every 20 epochs interval.

4.1.2. Pruning and Retraining Configuration

We trim a small but fixed percentage of filters in every epoch and perform retraining for one epoch. First few epochs are used to iteratively prune and retrain until the FLOPs target is achieved and then the remaining epochs are used only for fine-tuning. The pruned model uses the same hyperparameters as the base model to complete retraining and fine-tuning. We prune 1% of the filters of the entire network in every epoch for CIFAR experiments. For the ILSVRC-12 dataset, we retrain and fine-tune ResNet50 and MobileNetV2 for 100 and 150 epochs respectively, compared to 300 for CIFAR datasets. So we prune large number of filters, specifically 2.5% filters from the model every epoch for the ILSVRC-12 dataset. This ensures that the final pruned model gets sufficient epochs for fine tuning. FPSL adaptively prunes different amount of filters from different layers while following iterative prune and retrain. We retain at least two filters in each layer to avoid severe performance degradation or layer collapse at any time during pruning. We utilize a step scheduler with a initial learning rate of 0.1 divided by 10 in every 30 epochs for ResNet50. For multi-branch networks, we prune all convolutional layers except the last one of each residual block to avoid structural inconsistency due to pruning. FPSL can also be applied to the base model without pretraining. In these cases, we prune after the model has been trained for a few number of epochs, specifically 5% of the total epochs in all experiments.

4.2. Results and Analysis

Here we compare the performance of FPSL with several state of the art (SOTA) pruning methods including LRMF [56], MFP [12], Filter Sketch [26], LFPC [8], HRank [28], ASFP [9], SFP [10], FPGM [11], GAL [30]. Although FPSL is an extremely fast feature-independent filter pruning method but this comparison includes all types of recent filter pruning methods irrespective of whether it is feature dependent or optimization based.

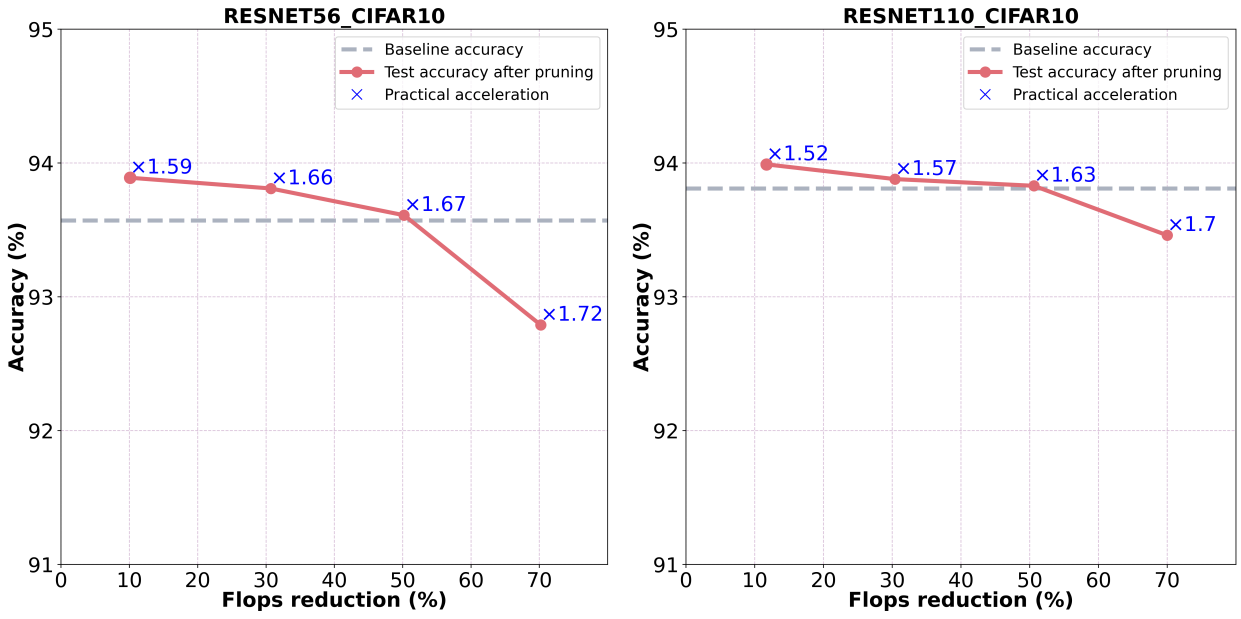

Fig. 5. Performance of FPSL on CIFAR10 with ResNet56 (left) and ResNet110 (right), at varying FLOPs reduction. FPSL successfully maintains the baseline accuracy while reducing FLOPs by 50% for both ResNet56 and ResNet110. FPSL offers a practical inference acceleration of 1.67 times for ResNet56 and 1.63 times for ResNet110 compared to the corresponding unpruned model, without any compromise in performance.

We demonstrate the superiority of FPSL over existing methods in terms of performance in this section. We also observe the performance of the pruned model compared to the unpruned. Here, baseline accuracy, unpruned accuracy, and the original network's accuracy these three terminologies indicate the same thing, i.e., the accuracy of the model when no filters are pruned.

Table 1. Pruning Results of ResNet56 on CIFAR10.

Method	Unpruned Top1 Acc (%)	Pruned Top1 Acc (%)	Acc. drop (%)	FLOPs rdcn (%)
CP [13]	92.8	91.8	1.00	50
DCP [63]	93.8	93.49	0.31	50
HRank [28]	93.26	93.17	0.09	50
ASFP [9]	93.59	93.12	0.47	52.6
SFP [10]	93.59	93.35	0.24	52.6
MFP [12]	93.59	93.56	0.03	52.6
LRMF [56]	93.59	93.25	0.34	52.6
FPGM [11]	93.59	93.26	0.33	52.6
LFPC [8]	93.59	93.24	0.35	52.9
FPSL	**93.57**	**93.42**	**0.15**	**53.4**

4.2.1. ResNet on CIFAR10

We apply FPSL on multi-branch networks like ResNet56, ResNet110 while using CIFAR10 dataset and observe the pruning results that are tabulated in Table 1 and Table 2 respectively. FPSL 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 1.

Similarly, Table 2 shows that FPSL 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 [56], LFPC [8], HRank [28] and many others. Moreover, FPSL has shown significant better performance than the existing filter norm based methods like PFEC [23], SFP [10], ASFP [9], e.g. - SFP decreases 52.3% FLOPs with the accuracy drop of 0.78% whereas FPSL provides 60.8% FLOPs reduction with a very small amount of 0.07% accuracy drop. We also observe the performance of the pruned model while varying the flops reduction percentage. FPSL reduces 50% FLOPs without degrading the model performance both for ResNet56_CIFAR10 (Fig. 5 (left)) and ResNet110_CIFAR10 (Fig. 5 (right)). Fig. 5 also highlights that reducing half of the FLPOs leads to a practical inference acceleration of 1.67 times for ResNet56 and 1.63 times for ResNet110 compared to the corresponding unpruned models.

Table 2. Pruning Results of ResNet110 on CIFAR10.

Method	Unpruned Top1 Acc (%)	Pruned Top1 Acc (%)	Acc. drop (%)	FLOPs rdcn (%)
PFEC [23]	93.53	93.3	0.23	38.6
GAL [30]	93.5	92.74	0.76	48.5
SFP [10]	93.68	92.90	0.78	52.3
ASFP [9]	93.68	93.1	0.58	52.3
MFP [12]	93.68	93.31	0.37	52.3
FPGM [11]	93.68	93.74	-0.06	52.3
LRMF [56]	93.68	93.88	-0.20	52.3
HRank [28]	93.5	93.36	0.14	58.2
LFPC [8]	93.68	93.07	0.61	60.3
FPSL	**93.81**	**93.74**	**0.07**	**60.8**

4.2.2. VGG on CIFAR datasets

We also apply our method on single branch networks like VGG16 which has 13 convolutional layers and 3 fully connected layers. Table 3 indicates that FPSL improves over baseline accuracy by 0.07% for CIFAR10 while lowering computational burden by 53.6%. Furthermore, even for 100 class

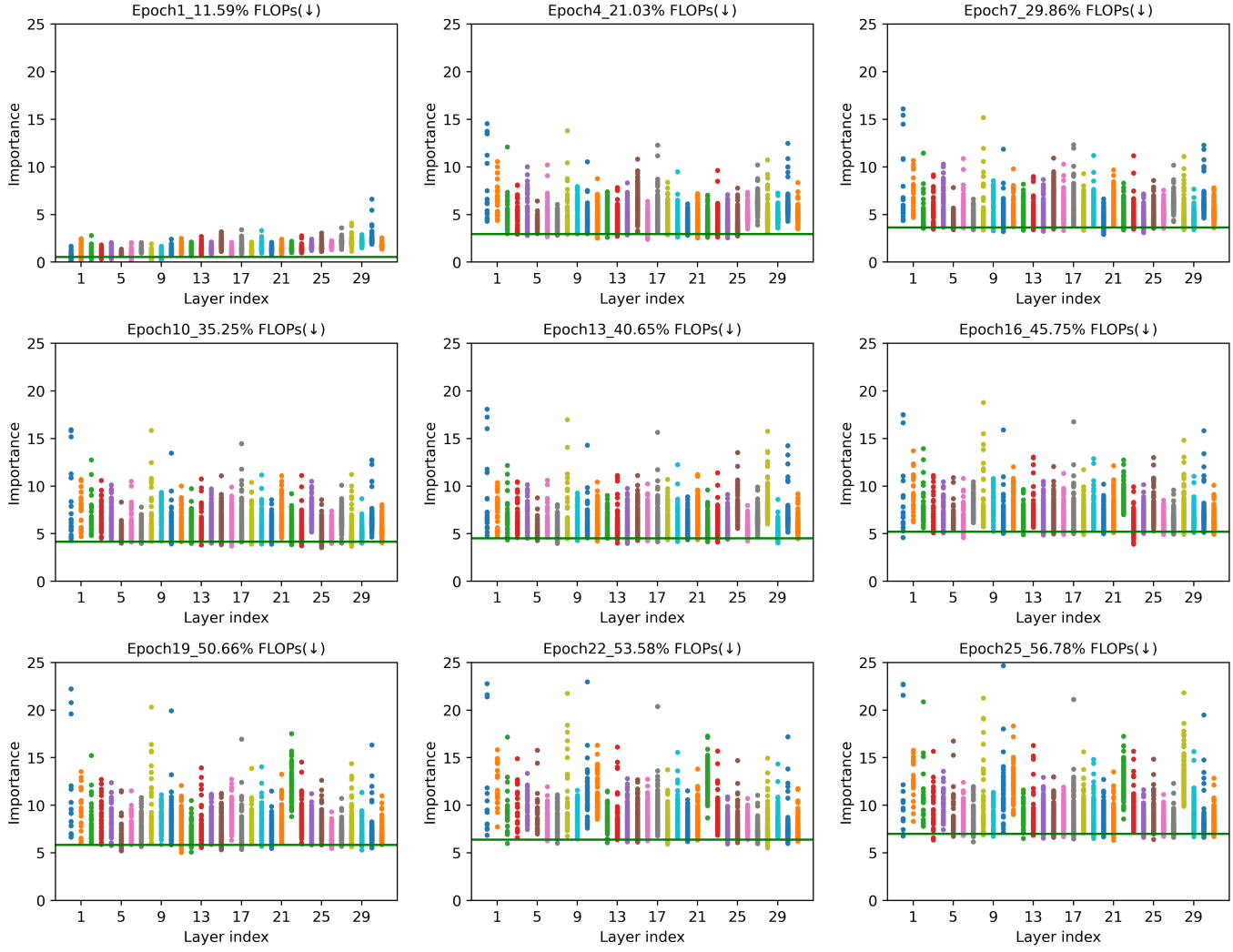

Fig. 6. Filter importance of all the filters for ResNet50_ImageNet configuration. Horizontal green line indicates the threshold, filters with importance score lesser than the threshold is pruned. Epoch number and the reduction in total FLOPs compared to base model is highlighted on the top of each subplot.

classification with the CIFAR100 [22] dataset, FPSL does not degrade the model performance of VGG16 while saving more than 50% FLOPs. Under the similar FLOPs constraints, FPSL outperforms the existing methods like HRank [28] which decreases the classification accuracy by 0.53% whereas FPSL increases the classification accuracy by 0.07% for CIFAR10.

Table 3. Pruning Results of VGG16 on CIFAR datasets.

Model_Dataset	Method	Unpruned Top1 Acc (%)	Pruned Top1 Acc (%)	Acc. drop (%)	FLOPs rdcn (%)
VGG16_CIFAR10	PFEC [23]	93.25	93.4	-0.15	34.2
	GAL [30]	93.96	93.42	0.54	45.2
	HRank [28]	93.96	93.43	0.53	53.5
	FPSL	**93.9**	**93.97**	**-0.07**	**53.6**
VGG16_CIFAR100	GFI-AP [39]	73.97	73.68	0.29	53.5
	FPSL	**74.29**	**74.3**	**-0.01**	**53.6**

4.2.3. ResNet50 on ImageNet

We evaluate the performance of FPSL when pruning ResNet50, which is utilized for the ImageNet dataset. It is difficult to prune ResNet50 on ImageNet without performance degradation since it is a large-scale diverse dataset having 1000 classes. However, ResNet-50 is often used to evaluate pruning algorithms due to its strong generalization capabilities to other architectures. We observe from Table 4 that FPSL reduces 42.7% FLOPs but still maintains the baseline top-1 accuracy for ResNet50_ImageNet configuration. Table 4 also shows that FPSL achieves 54.5% FLOPs reduction while decreasing the top-1 accuracy by 0.94% whereas existing methods like FPGM [11] decrease the top-1 by 2.02% and MFP [12] by 1.29% even with lesser than 54.5% FLOPs reduction. As seen in Fig. 1, FPSL consistently outperforms SOTA and delivers superior top-1 accuracy for varying FLOPs reduction rates. This highlights the superiority of FPSL over other pruning algorithms even for challenging datasets like ImageNet. Fig. 6 shows the variation in the filter importance and threshold while performing iterative pruning and retraining. It indicates that as the pruning increases with iteration, more FLOPs are reduced over time. The importance score and threshold increase with time as fewer filters in the retained model become responsible for performing the same classification task.

Table 4. 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 [17]	4.09	76.15%	74.18%	1.97%	92.96%	91.91%	1.05%	2.82	31.90%
CP [13]	4.09	76.15%	72.30%	3.85%	92.96%	90.80%	2.16%	2.73	34.10%
FilterSketch-0.7 [26]	4.09	76.13%	75.22%	0.91%	92.86%	92.41%	0.45%	2.64	35.50%
SFP [10]	4.09	76.15%	74.61%	1.54%	92.87%	92.06%	0.81%	2.39	41.80%
Epruner-0.71 [27]	4.13	76.01%	74.95%	1.06%	92.96%	92.36%	0.60%	2.37	42.60%
MFP 30% [12]	4.09	76.15%	75.67%	0.48%	92.87%	92.81%	0.06%	2.36	42.20%
FPSL 0.42	**4.12**	**76.15%**	**76.15%**	**0.00%**	**92.87%**	**92.92%**	**-0.05%**	**2.36**	**42.70%**
GAL [30]	4.09	76.15%	71.95%	4.20%	92.96%	90.79%	2.17%	2.33	43.70%
SSS-32 [17]	4.09	76.12%	71.82%	4.30%	92.86%	90.79%	2.07%	2.33	43.70%
HRank [28]	4.09	76.15%	75.01%	1.14%	92.96%	92.33%	0.63%	2.3	43.90%
FilterSketch-0.6 [26]	4.09	76.13%	74.68%	1.45%	92.86%	92.17%	0.69%	2.23	45.50%
White-Box [57]	4.09	76.15%	75.32%	0.83%	92.96%	92.43%	0.53%	2.22	45.60%
ABP [48]	3.89	75.88%	74.80%	1.08%	92.76%	92.36%	0.40%	2.21	42.80%
GFI-AP [39]	3.89	75.95%	74.61%	1.34%	92.89%	92.01%	0.88%	2.01	48.33%
FPGM [11]	4.09	76.15%	74.13%	2.02%	92.96%	92.87%	0.09%	1.9	53.50%
MFP 40% [12]	4.09	76.15%	74.86%	1.29%	92.87%	92.43%	0.44%	1.9	53.50%
FPSL 0.54	**4.12**	**76.15%**	**75.21%**	**0.94%**	**92.87%**	**92.44%**	**0.43%**	**1.88**	**54.50%**
RRBP [62]	4.09	76.15%	73.00%	3.15%	92.96%	91.00%	1.96%	1.86	54.50%
GAL [30]	4.09	76.15%	71.80%	4.35%	92.96%	90.82%	2.14%	1.84	55.60%
ThiNet 50 [37]	3.86	75.30%	72.03%	3.27%	92.20%	90.99%	1.21%	1.71	55.80%
HRel [43]	4.12	76.15%	74.54%	1.61%	92.87%	92.12%	0.75%	1.69	58.90%
LFPC [8]	4.09	76.15%	74.18%	1.97%	92.96%	91.92%	1.04%	1.61	60.80%
HRank [28]	4.09	76.15%	71.98%	4.17%	92.96%	91.01%	1.95%	1.55	62.60%
FPSL 0.62	**4.12**	**76.15%**	**74.55%**	**1.60%**	**92.87%**	**92.06%**	**0.81%**	**1.54**	**62.60%**

4.2.4. 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 ImageNet dataset. However FPSL can still reduce 20% FLOPs with only 0.52% top-1 accuracy drop for MobileNetV2_ImageNet configuration as shown in Table 5. Moreover, it provides a 2.1% improvement in terms of top-1 accuracy drop over competing methods like DCP [63] while reducing more FLOPs (45.8%).

Table 5. Pruning Results of MobileNetV2 on ImageNet.

Method	Unpruned Top1 Acc (%)	Pruned Top1 Acc (%)	Acc. drop (%)	FLOPs rdcn (%)
FPSL	**70.36**	**69.84**	**0.52**	**19.5**
DCP [63]	70.11	64.22	5.89	44.8
FPSL	**70.36**	**66.59**	**3.77**	**45.8**

4.2.5. Parameters reduction while applying FPSL

In this subsection, we examine the reduction in parameters achieved through FPSL while targeting a specific percentage of FLOPs (computational operations) reduction. It is important to note that FLOPs, which determine the computational complexity, typically surpass the number of parameters in deep CNN models by a factor of 10 to 200 due to the increased number of convolutional operations. For instance, the unpruned ResNet50 model has 25.56 million parameters but requires 4.12 billion FLOPs for ImageNet classification. Table 6 highlights the relationship between the desired FLOPs reduction (%), the corresponding decrease in the number of parameters, and the percentage of parameters eliminated using FPSL. In this table, the abbreviations C10 and C100 denote CIFAR10 and CIFAR100 datasets, respectively. We observe that employing FPSL results in a substantial 66.9% reduction in network parameters, equating to the elimination of 10.23 million parameters, when targeting a 36.3% decrease in FLOPs for the CIFAR100_VGG16 configuration.

Table 6. Parameters and FLOPs reduction while applying FPSL

Data_Arch	Params rdcn (M)	Params rdcn (%)	FLOPs rdcn (M)	FLOPs rdcn (%)	Acc. drop (%)
C10_ResNet56	0.36	42.6	68.1	53.4	0.15
C10_ResNet110	0.89	51.6	156.7	60.8	0.07
C10_VGG16	13.09	86	168.7	53.6	**-0.07**
C100_VGG16	10.23	66.9	114.6	36.3	**-0.01**
ImageNet_ResNet50	6.98	27.3	1759	42.7	**0**
ImageNet_ResNet50	10.58	41.4	2246	54.5	0.94
ImageNet_ResNet50	13.22	51.7	2579	62.6	1.6
ImageNet_MobileNetV2	0.5	14.2	65.8	20.5	0.52

4.3. More Explorations

4.3.1. FPSL using only current or subsequent layer

Here 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. FPSL-C indicates when the contribution of a filter belonging to the current layer is only used to determine the filter importance. FPSL-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. 7 shows that FPSL, which uses both

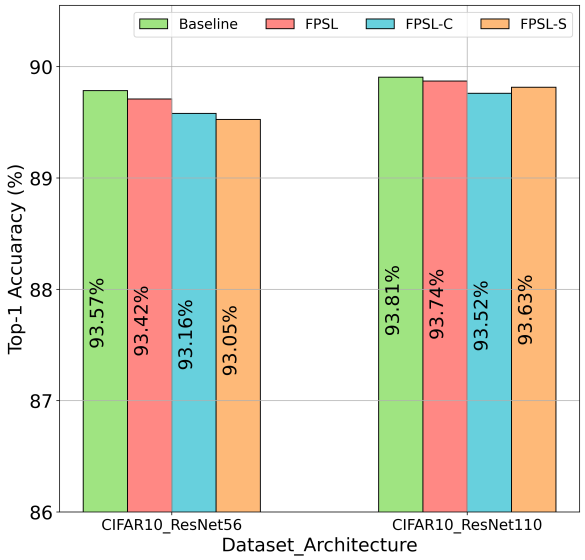

Fig. 7. Performance comparison between FPSL, FPSL_C and FPSL_S for ResNet56 and ResNet110 for CIFAR10 classification. FPSL_C indicates when only the current layer filter information is used for filter importance. similarly, FPSL_S indicates when only the succeeding layer filter information is used for filter importance. FPSL outperforms both FPSL-C and FPSL-S in all cases as it uses both the current and the succeeding layer filter information to compute filter importance.

the current and the succeeding layer filter contribution, outperforms FPSL-C and FPSL-S for both CIFAR10_ResNet56 and CIFAR10_ResNet110 configuration. FPSL provides 0.26% improvement over FPSL-C and 0.37% improvement over FPSL-S while pruning 53% FLOPs for CIFAR10_ResNet56 configuration. Similarly, when we reduce 61% FLOPs from ResNet110 while performing CIFAR10 classification, we observe that FPSL-S provides better results than FPSL-C. However FPSL outperforms FPSL-C and FPSL-S also in this case.

4.3.2. Computation time to determine filter importance

We also compare the time required to compute the filter importance using a feature independent method (FPSL) with a feature dependent method (GFI-AP[39]) in Table 7. We have used a system with Intel® Xeon(R) Silver 4214 CPU @ 2.20GHz × 48 processor, 128GB RAM and a NVIDIA RTX 2080 TI graphics driver to perform this experiment for both methods. We find that GFI-AP requires 311.6 secs. and 956.2 secs. to compute the filter importance of all the filters belong to ResNet56 and ResNet110 while performing CIFAR10 classification. Whereas, FPSL takes less than 2 secs. to compute the filter importance for both configuration. This shows that FPSL is exceptionally fast irrespective of the network size. For deeper models like ResNet110, the superiority of FPSL becomes more prominent as feature dependent methods require enormous time to compute and process the feature maps of the entire network for all training examples.

Table 7. Computation time to compute filter importance using GFI-AP (feature dependent) and FPSL (feature independent) method on ResNet56 and ResNet110 while performing CIFAR10 classification

Arch_Data	Time (Sec.)	
	GFI-AP[39]	FPSL
C10_R56	311.6	**1.3**
C10_R110	956.2	**1.85**

4.3.3. Effect of pretraining on FPSL

Finally we apply FPSL on models which are not pretrained to observe the effects of pretraining on pruning. 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. We refer to this method as FPSL-NP indicating FPSL applied on a model which is not pretrained.

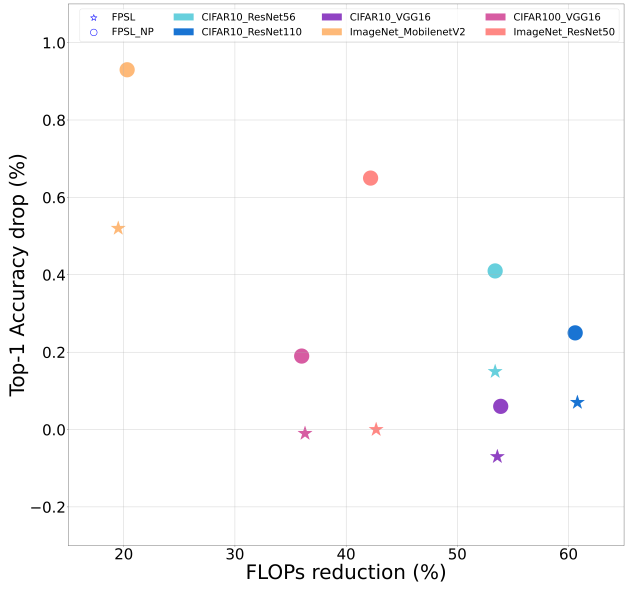

Fig. 8. Performance comparison between FPSL and FPSL_NP for different datasets and architectures. FPSL_NP means when FPSL applied to a base model which is Not Pretrained. Top-1 accuracy drop for both FPSL and FPSL_NP are very close to each other for similar amount of FLOPs reduction for both VGG and Residual nets while working with CIFAR datasets

We observe from Fig. 8 that the top-1 accuracy drop for FPSL and FPSL-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 FPSL whereas it is 0.25% for FPSL-NP while reducing more than 60% FLOPs in both cases for CIFAR10_ResNet110 configuration. So FPSL can efficiently prune the base models even if they are not pretrained. However the difference between FPSL and FPSL-NP increases for large diverse datasets like ImageNet. For example, FPSL provides no drop in top-1 accuracy but FPSL-NP provides a 0.65% top-1 accuracy drop while reducing 43% FLOPs for ResNet50_ImageNet configuration and the difference between FPSL and FPSL-NP becomes 0.41% with 20% FLOPs reduction while utilizing MobileNetV2 for ImageNet classification.

It indicates that the impact of pretraining becomes prominent when the dataset is challenging and the base model is relatively small.

5. Limitations

While FPSL offers valuable benefits like deploying compact models on resource-constrained systems, iterative pruning and retraining for improved decision-making, and enhanced model interpretability, it is also important to discuss its limitations. Firstly, FPSL reduces computational complexity during inference but requires additional computational resources for the pruning and retraining stages. Secondly, although we chose the same pruning fraction per epoch for simplicity, further improvement in the performance can be achieved by selecting different pruning fractions per epoch based on the training stage. Lastly, while our method has been effective for pruning convolutional neural networks, it cannot be directly applied to transformer-based models. Further research is needed to explore how the principles of FPSL can be extended and utilized for attention-based models, such as transformers.

6. Conclusions

To conclude, we propose FPSL that considers how pruning a filter impact the feature maps in the immediate next layer. We find that a filter and its corresponding channel of all the filters in the immediate next layer both are directly responsible for generating the feature maps of the next layer. Hence, FPSL considers both contributions when estimating the importance of a filter, as opposed to previous methods that rely only on the current layer's weights. FPSL does not require any feature map information to compute importance. Thus, our fast algorithm allows us to 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. Furthermore, unlike existing methods, FPSL does not require layer-wise retraining, thorough hyperparameter search for fine-tuning, or human intervention to specify per layer filter pruning percentage. Rather, it automatically prunes filters until the target FLOPs is achieved. FPSL outperforms the SOTA filter pruning methods. For instance, FPSL 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. The capability of FPSL to facilitate the deployment of advanced deep learning models on low-power devices demonstrates its potential to drive several positive social impacts. The increased accessibility of AI technologies can stimulate innovation and promote development in various sectors. FPSL enables faster inference in pruned models, enhancing real-time performance in domains such as healthcare, autonomous vehicles, and smart systems, thereby improving timely decision-making. Moreover, FPSL contributes to environmental sustainability and energy efficiency by reducing energy consumption and the carbon footprint associated with AI applications.

References

[1] Beyer, L., Zhai, X., Royer, A., Markeeva, L., Anil, R., Kolesnikov, A., 2022. Knowledge distillation: A good teacher is patient and consistent, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10925–10934.

[2] Dai, X., Yin, H., Jha, N.K., 2019. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. IEEE Transactions on Computers 68, 1487–1497.

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

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

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

[6] Hayashi, K., Yamaguchi, T., Sugawara, Y., Maeda, S.i., 2019. Exploring unexplored tensor network decompositions for convolutional neural networks. Advances in Neural Information Processing Systems 32.

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

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

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

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

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

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

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

[14] Hinton, G., Vinyals, O., Dean, J., et al., 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 2.

[15] Hu, H., Peng, R., Tai, Y.W., Tang, C.K., 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 .

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

[17] Huang, Z., Wang, N., 2018. Data-driven sparse structure selection for deep neural networks, in: Proceedings of the European conference on computer vision (ECCV), pp. 304–320.

[18] Idelbayev, Y., Carreira-Perpinán, M.A., 2020. 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, pp. 8049–8059.

[19] Ji, Y., Liang, L., Deng, L., Zhang, Y., Zhang, Y., Xie, Y., 2018. Tetris: Tile-matching the tremendous irregular sparsity. Advances in neural information processing systems 31.

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

[21] Kim, H., Khan, M.U.K., Kyung, C.M., 2019. Efficient neural network compression, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 12569–12577.

[22] Krizhevsky, A., Hinton, G., et al., 2009. Learning multiple layers of features from tiny images .

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

[24] Li, Y., Gu, S., Mayer, C., Gool, L.V., Timofte, R., 2020. Group spar-

sity: The hinge between filter pruning and decomposition for network compression, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 8018–8027.

[25] Li, Y., Pan, Y., Yao, T., Mei, T., 2022. Comprehending and ordering semantics for image captioning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 17990–17999.

[26] Lin, M., Cao, L., Li, S., Ye, Q., Tian, Y., Liu, J., Tian, Q., Ji, R., 2021a. Filter sketch for network pruning. IEEE Transactions on Neural Networks and Learning Systems .

[27] Lin, M., Ji, R., Li, S., Wang, Y., Wu, Y., Huang, F., Ye, Q., 2021b. Network pruning using adaptive exemplar filters. IEEE Transactions on Neural Networks and Learning Systems .

[28] Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L., 2020. Hrank: Filter pruning using high-rank feature map, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1529–1538.

[29] Lin, M., Zhang, Y., Li, Y., Chen, B., Chao, F., Wang, M., Li, S., Tian, Y., Ji, R., 2022. 1xn pattern for pruning convolutional neural networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 45, 3999–4008.

[30] Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., Doermann, D., 2019. Towards optimal structured cnn pruning via generative adversarial learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2790–2799.

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

[32] Liu, Z., Luo, W., Wu, B., Yang, X., Liu, W., Cheng, K.T., 2020. Bi-real net: Binarizing deep network towards real-network performance. International Journal of Computer Vision 128, 202–219.

[33] Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T., Sun, J., 2019. Metapruning: Meta learning for automatic neural network channel pruning, in: Proceedings of the IEEE/CVF international conference on computer vision, pp. 3296–3305.

[34] Liu, Z., Xu, J., Peng, X., Xiong, R., 2018. Frequency-domain dynamic pruning for convolutional neural networks, in: Advances in Neural Information Processing Systems, pp. 1043–1053.

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

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

[37] Luo, J.H., Zhang, H., Zhou, H.Y., Xie, C.W., Wu, J., Lin, W., 2018. Thinet: pruning cnn filters for a thinner net. IEEE transactions on pattern analysis and machine intelligence 41, 2525–2538.

[38] Maile, K., Rachelson, E., Luga, H., Wilson, D.G., 2022. When, where, and how to add new neurons to anns, in: International Conference on Automated Machine Learning, PMLR. pp. 18–1.

[39] Mondal, M., Das, B., Roy, S.D., Singh, P., Lall, B., Joshi, S.D., 2022. Adaptive cnn filter pruning using global importance metric. Computer Vision and Image Understanding 222, 103511.

[40] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32.

[41] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al., 2015. Imagenet large scale visual recognition challenge. International journal of computer vision 115, 211–252.

[42] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4510–4520.

[43] Sarvani, C., Ghorai, M., Dubey, S.R., Basha, S.S., 2022. Hrel: Filter pruning based on high relevance between activation maps and class labels. Neural Networks 147, 186–197.

[44] Shi, J., Xu, J., Tasaka, K., Chen, Z., 2020. Sasl: saliency-adaptive sparsity learning for neural network acceleration. IEEE Transactions on Circuits and Systems for Video Technology 31, 2008–2019.

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

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

[47] Tang, Y., Wang, Y., Xu, Y., Deng, Y., Xu, C., Tao, D., Xu, C., 2021. Manifold regularized dynamic network pruning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5018–5028.

[48] Tian, G., Sun, Y., Liu, Y., Zeng, X., Wang, M., Liu, Y., Zhang, J., Chen, J., 2021. Adding before pruning: Sparse filter fusion for deep convolutional neural networks via auxiliary attention. IEEE Transactions on Neural Networks and Learning Systems .

[49] Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S., 2019. Haq: Hardware-aware automated quantization with mixed precision, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8612–8620.

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

[51] Xie, X., Cheng, G., Wang, J., Yao, X., Han, J., 2021. Oriented r-cnn for object detection, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 3520–3529.

[52] Yang, Y., Zhuang, Y., Pan, Y., 2021. Multiple knowledge representation for big data artificial intelligence: framework, applications, and case studies. Frontiers of Information Technology & Electronic Engineering 22, 1551–1558.

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

[54] Yu, R., Li, A., Chen, C.F., Lai, J.H., Morariu, V.I., Han, X., Gao, M., Lin, C.Y., Davis, L.S., 2018. Nisp: Pruning networks using neuron importance score propagation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9194–9203.

[55] Zeng, L., Tian, X., 2018. Accelerating convolutional neural networks by removing interspatial and interkernel redundancies. IEEE transactions on cybernetics 50, 452–464.

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

[57] Zhang, Y., Lin, M., Lin, C.W., Chen, J., Wu, Y., Tian, Y., Ji, R., 2022a. Carrying out cnn channel pruning in a white box. IEEE Transactions on Neural Networks and Learning Systems .

[58] Zhang, Y., Lin, M., Lin, Z., Luo, Y., Li, K., Chao, F., Wu, Y., Ji, R., 2022b. Learning best combination for efficient n: M sparsity. Advances in Neural Information Processing Systems 35, 941–953.

[59] Zhang, Y., Luo, Y., Lin, M., Zhong, Y., Xie, J., Chao, F., Ji, R., 2023. Bi-directional masks for efficient n: M sparse training. arXiv preprint arXiv:2302.06058 .

[60] Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., Li, H., 2021. Learning n: m fine-grained structured sparse neural networks from scratch. arXiv preprint arXiv:2102.04010 .

[61] Zhou, Y., Yen, G.G., Yi, Z., 2019a. A knee-guided evolutionary algorithm for compressing deep neural networks. IEEE transactions on cybernetics 51, 1626–1638.

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

[63] Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J., 2018. Discrimination-aware channel pruning for deep neural networks, in: Advances in Neural Information Processing Systems, pp. 875–886.