01 Mar 2022

Heuristic-Based Automatic Pruning of Deep Neural Networks

Tejalal Choudhary

Vipul Mishra

Anurag Goswami

Jagannathan Sarangapani
Missouri University of Science and Technology, sarangap@mst.edu

Recommended Citation

T. Choudhary et al., "Heuristic-Based Automatic Pruning of Deep Neural Networks," *Neural Computing and Applications*, vol. 34, no. 6, pp. 4889 - 4903, Springer, Mar 2022.
The definitive version is available at https://doi.org/10.1007/s00521-021-06679-z

Neural Computing and Applications (2022) 34:4889–4903
https://doi.org/10.1007/s00521-021-06679-z

ORIGINAL ARTICLE

Heuristic-based automatic pruning of deep neural networks

Tejalal Choudhary[1] · Vipul Mishra[1] · Anurag Goswami[1] · Jagannathan Sarangapani[2]

Received: 20 January 2021 / Accepted: 27 October 2021 / Published online: 10 January 2022

Abstract
The performance of a deep neural network (deep NN) is dependent upon a significant number of weight parameters that need to be trained which is a computational bottleneck. The growing trend of deeper architectures poses a restriction on the training and inference scheme on resource-constrained devices. Pruning is an important method for removing the deep NN's unimportant parameters and making their deployment easier on resource-constrained devices for practical applications. In this paper, we proposed a heuristics-based novel filter pruning method to automatically identify and prune the unimportant filters and make the inference process faster on devices with limited resource availability. The selection of the unimportant filters is made by a novel pruning estimator (γ). The proposed method is tested on various convolutional architectures AlexNet, VGG16, ResNet34, and datasets CIFAR10, CIFAR100, and ImageNet. The experimental results on a large-scale ImageNet dataset show that the FLOPs of the VGG16 can be reduced up to 77.47%, achieving $\approx 5x$ inference speedup. The FLOPs of a more popular ResNet34 model are reduced by 41.94% while retaining competitive performance compared to other state-of-the-art methods.

Keywords Deep neural network · Efficient inference · Convolutional neural network · Model compression and acceleration · Filter pruning

1 Introduction

Deep neural networks (DNNs) have been applied in a wide variety of domains such as computer vision, machine translation, natural language processing and have achieved state-of-the-art performance in various applications such as classification [33, 47] object detection [13, 37, 42] image generation [15], speech recognition [16], and segmentation [2] to name a few. The generalization, ability to perform well on the unseen data [52] and convergence ability of DNN comes from their millions of weight parameters and billions of floating-point operations (FLOPs) that has helped DNNs in achieving superior performance over time. Over the years, neural network (NN) architectures have become deeper with an increasing number of layers and wider meaning more nodes or filters at each layer. Convolutional neural network (CNN) is one of the popular variants of the DNN. Figure 1 shows the architecture of the VGG16 convolutional architecture. Recently, researchers have successfully trained a network with more than 150 layers and achieved improved performance on a wide variety of tasks such as classification, detection, and segmentation [20].

Despite improvements in their performance [20, 47], however, their usage on resource-constrained devices like mobile phones, wearables, and other edge-devices has hindered [28, 36]. For practical applications, it has become a necessity to bring the capabilities of DNNs in devices with limited resources [17, 50]. Inference process of trained networks on devices with inadequate resources leads to the following challenges: (1) for instance in CNN, performing convolutional operation requires a lot of computational power [7, 57], and (2) the battery power or

✉ Vipul Mishra
vipul.mishra@bennett.edu.in

Tejalal Choudhary
tejalal.choudhary@gmail.com

Anurag Goswami
anurag.goswami@bennett.edu.in

Jagannathan Sarangapani
sarangap@mst.edu

[1] Bennett University, Greater Noida 201310, India

[2] Missouri University of Science and Technology, Rolla, MO 65409, USA

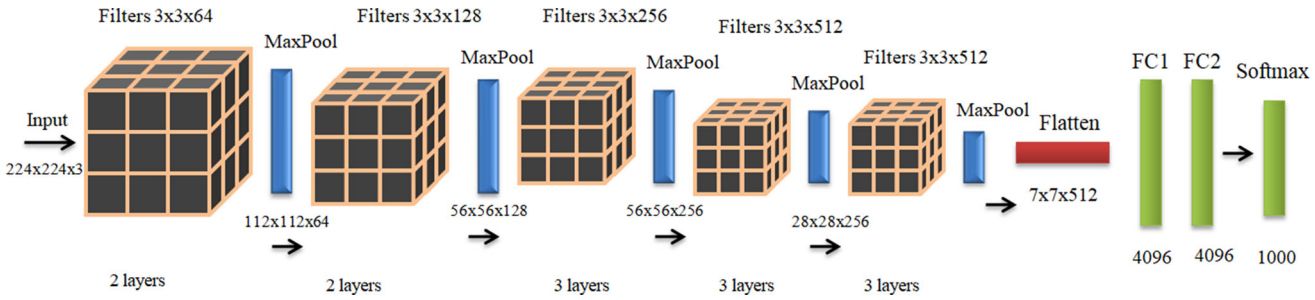

Fig. 1 An example of VGG16 CNN

energy needed [25] to run CNN which performs a lot of FLOPs operations during inference.

Therefore, model compression and acceleration methods have attracted significant attention of the researchers from deep learning community [6, 10, 27, 53]. Alternatively, a large number of unimportant weight parameters can be removed without affecting the performance of the model [22, 48] at the same time reducing the FLOPs, though it is challenging [9]. In summary, significant research efforts have been devoted to making DNNs inference-friendly so that they can be deployed on devices with limited resources [35, 40].

Out of the many compression and acceleration techniques such as low-rank approximation [30, 46, 55], quantization and binarization [3, 27, 49], knowledge distillation [4, 24], pruning has evolved as one of the important methods to improve DNNs efficiency for limited budget applications and devices [39, 48]. In pruning, the less important parameters of the model are pruned; either set to zero or completely removed from the network based on the type of pruning performed. From a DNN, individual weight connections, a complete node, filters, or layers can be pruned and referred to as weight pruning [18, 19, 34], node pruning [44], filter pruning [1, 21, 40], and layer pruning [5], respectively.

Pruning can be categorized as structured or unstructured. For example, pruning individual weight connections or nodes is unstructured as it makes the resulting architecture sparse [18]. On the other hand, the complete filter or layer's pruning is structured [35, 39]. Pruning also reduces the well-known over-fitting problem of the DNNs [12, 14]. For a comprehensive study on other compression and acceleration techniques, the readers are suggested to refer to recent surveys [8, 9].

This paper aims to propose a novel heuristics-based automatic filter pruning method for accelerating the performance of the DNNs. In particular, the objectives are to analyze whether the inference-performance of the DNNs can be improved based on the reduction of the time-consuming FLOPs while keeping the accuracy loss within the acceptable range; and to investigate whether the proposed method can be generalized for different convolutional architectures.

The earlier research [22, 35, 40] has shown that many convolutional layer filters do not contain useful information after training, hence become unimportant and can be pruned. Instead of utilizing a trained model directly on the target device, we first propose to prune the unimportant filters, re-train the pruned model, and finally deploy a pruned and fine-tuned model on the target device as shown in Fig. 2. Unlike other techniques [1, 26, 35], our proposed method eliminates the need for any manual tuning to find the threshold for each layer. Instead, the number of filters that could be pruned from each layer is determined automatically.

The main contributions of the paper include: (1) a novel heuristics-based automatic filter pruning method to identify and prune unimportant filters from the trained CNN without the need for human intervention during the pruning process; (2) introduction of a novel pruning estimator γ wherein the filters whose absolute sum is less than γ are treated as unimportant and pruned layer-wise. (3) addition of an additional control parameter (β) to handle the situations where the model is to be pruned more or less aggressively.

This paper has been organized into various sections. In Sect. 2, our proposed method is explained that will help to achieve our research objectives. Section 3 elucidates the details of the experiments performed with benchmarks convolutional architectures on various datasets. Section 4 includes the details of the experimental results, followed by the discussion. Finally, the conclusion and future directions are summarized in Sect. 5.

2 Methodology

This section discusses pruning as an optimization problem, followed by the details of the proposed method to automatically identify and prune unimportant filters from CNN. In CNN, a filter becomes unimportant if its removal does not affect the model performance after training. The

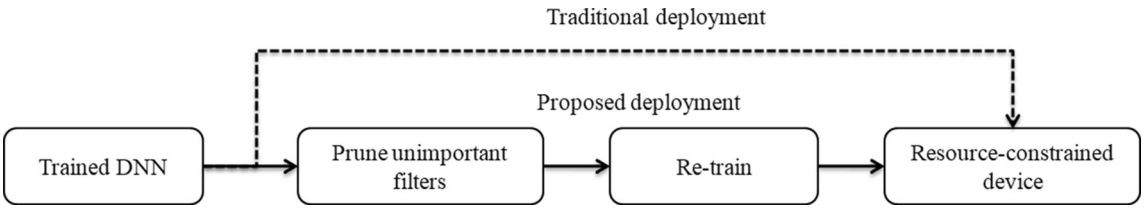

Fig. 2 Traditional (dotted line) and proposed (solid line) deployment scenario

performance of the model gets affected by the selection of proper filters for pruning. Therefore, it is necessary to prune the correct number of filters without affecting the model performance. If fewer filters are pruned, it will not help reduce the lesser importance filters. On the other hand, if the model is pruned aggressively, it becomes difficult for the pruned model to recover from the accuracy loss. Therefore, one of the main challenges in pruning is identifying candidate filters to prune, and the total number of filters that could be pruned from the model without compromising the performance. Pruning filters can be thought of as an optimization problem to maximize the performance of the model and reduce the resource requirements. In our case, the particular resource we consider minimizing is the FLOPs by pruning the unimportant filters and their respective feature maps.

Figure 3 visualizes the main steps of the proposed method. Our proposed method takes the trained model and pruning controller (β) as an input and performs pruning of less important filters for each of the model's convolutional layers. In Fig. 3, the pruned filters are highlighted with the red color. Solving the optimization problem can be considered as two main goals to be achieved. To solve this, we employed our proposed Algorithm 1. The proposed algorithm has two main parts pruning and fine-tuning. Pruning filters achieve the first goal, and the second goal is achieved by fine-tuning the pruned model. Algorithm 1 summarizes the entire filter pruning and fine-tuning steps of the proposed method. In Algorithm 1, β is the pruning controller, M is the original model, and M_{p} is the pruned model. Similarly, $W_m^{[l]}(\mathrm{old})$ and $W_k^{[l]}(\mathrm{new})$ represents the original and pruned weight of the lth convolutional layer of the model M and M_{p}, respectively. Overall, the proposed method can be categorized as filter selection, pruning lesser importance filters, and fine-tuning. In the next subsections, we have discussed each of these in detail.

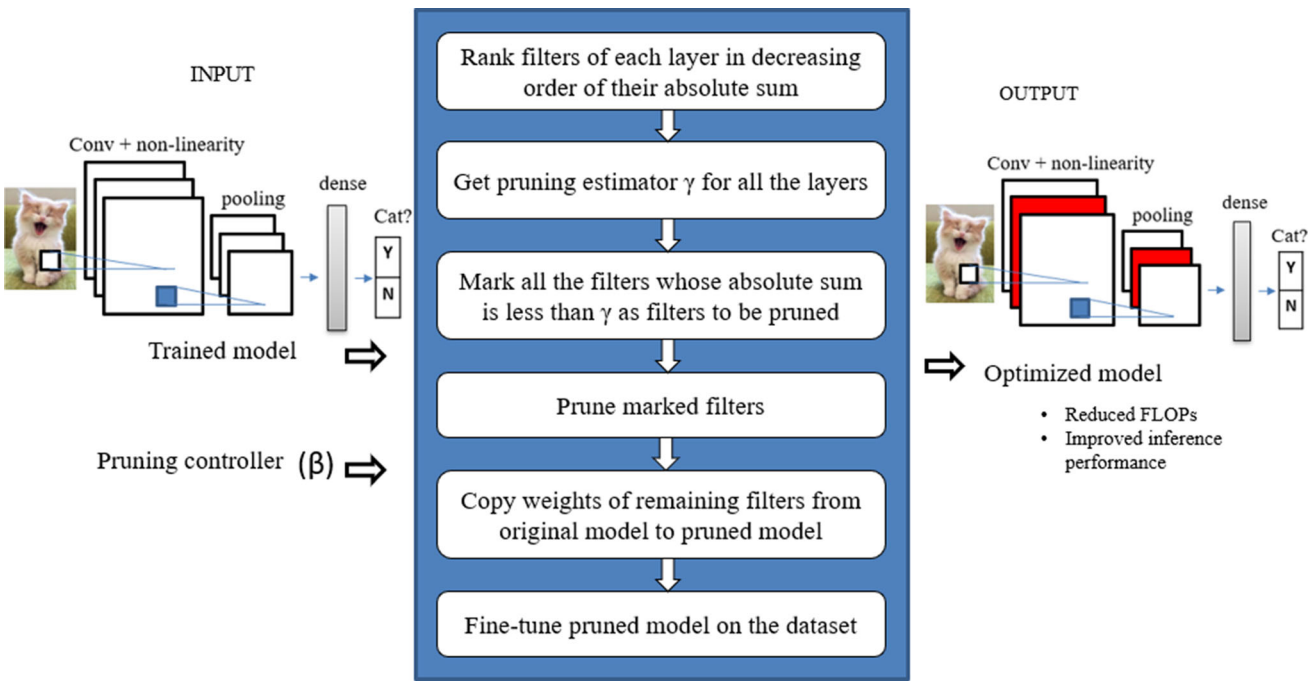

Fig. 3 Proposed filter pruning and fine-tuning framework

Algorithm 1: Heuristic-based filter pruning and fine-tuning

Input: Trained model M, dataset D, control parameter (β)
Output: Pruned and optimized model M_p with fewer filters
Initialization: Control parameter (β), fine-tuning parameters
// iterate through each convolutional layer and get the details of the filters to be pruned (line numbers 2-13)
foreach *convolutional layer* $l \in L$ **do**
  foreach *convolutional filter* $f^{(h\times w\times c)} \in F^{(h\times w\times c\times n)}$ **do**
    Calculate the absolute sum as
    $\tau_i = \sum_{i=1}^{c}\sum_{j=1}^{h}\sum_{k=1}^{w}|W_{i,j,k}|$
  end
  Rank the filters F in decreasing order based on their absolute sum
  Get the value of the pruning estimator (γ) for the layer l as
  $\gamma = \frac{\sum_{i=1}^{n}\tau_i}{\#filters} + \beta$
  if $\tau_i < \gamma$ **then**
    mark the respective filter in the layer l as pruning candidate
  end
end
// Perform one-shot pruning of all pruning candidate filters (line numbers (15-21)
Create new model architecture M_p with remaining filters
// Copy weights of the remaining filters from old model M to new model M_p (line numbers (15-21)
foreach *convolutional layer* $l \in L$ **do**
  $k \longleftarrow 0$
  foreach *filter* $m \in unprunedfilters$ **do**
    $W_k^{[l]}(new) = W_m^{[l]}(old)$
    $k \longleftarrow k+1$
  end
end
// Fine-tune pruned model (line numbers 22-25)
foreach *epoch* $e \in finetuneEpochs$ **do**
  Train M_p on the training set with fine-tuning parameters
  Test M_p on the test set with fine-tuning parameters
end
Return pruned and optimized model

2.1 Filter selection criteria

The selection of the filters for pruning is an important question to answer. In earlier research, it was found that small-magnitude weights are unimportant after training, and a substantial amount of weight parameters can be removed from the model [18], we extend the argument to convolutional layer filters. Filters with smaller absolute sum are unimportant and do not contain useful information; hence, pruning these filters will least affect the model performance. We proposed a novel heuristic pruning estimator (γ) to find the filters that could be pruned. In other words, γ is used to differentiate between the important and unimportant filters. The novel pruning estimator (γ) is defined as the mean absolute sum of all the filters in the layer (Eq. 5).

The research objective was to find an optimized model with the minimum number of convolutional filters while making the least compromise with the model performance. Let's assume M is the original model and M_p is the pruned model. The model M has C convolutional layers. The lth convolutional layer is represented by $l^{[k]}$, and $k \in \{1, 2, 3, \ldots, C\}$. The number of filters of the layer $l^{[k]}$ is n_s and generates the feature maps A_{map} that is used as input for the next layer. The set of filters of the model at layer $l^{[k]}$ is represented by $\mathcal{F}^{l[k]} = [f_1, f_2, f_3, \ldots, f_{n_s}]$. Further, let's assume the weights of the $l^{[k]}$ layer of the original model M is given by $W_m^{l[k]} = [w_1, w_2, w_3, \ldots, w_{n_s}]$. The weights of the $l^{[k]}$ layer of the pruned model M_p is given by $W_p^{l[k]} = [p_1, p_2, p_3, \ldots, p_{n_q}]$ where $n_s \neq n_q$. The accuracy of the original model M is given by α_m and the accuracy of the pruned model M_p is given by α_p. For a given dataset $\mathcal{D} = (x_i, y_i)_{i=1}^{N}$ and a pruning threshold level t given by the novel pruning estimators γ, the problem of pruning filters from the model was formulated as

$$\min_{\mathcal{F}} \quad \mathcal{L}(\mathcal{F}; \mathcal{D}) \tag{1}$$

$$= \min_{\mathcal{F}} \quad \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\mathcal{F}; (x_i, y_i)) \tag{2}$$

where $\mathcal{L}(.)$ is the standard cross entropy loss and $\mathcal{F}$ is the set of filters. Let's represent the absolute sum or $\ell 1$-norm of the filter f_i by τ. If $\tau \in \mathbb{R}$ then its norm is given by $|\tau|$, and defined as.

$$|\tau| = \sum_{i=1}^{c}\sum_{j=1}^{h}\sum_{m=1}^{w} |W_{i,j,m}| \tag{3}$$

For every τ following holds,
$|\tau| \geq 0$ for all $\tau \in \mathbb{R}$, and
$|\tau| = 0$ iff $\tau = 0$.

If $|\tau|$ is an integer greater than or equal to zero, then the sum of all the filters of the layer $l^{[k]}$ is given by

$$\tau_{\mathcal{F}} = \sum_{s=1}^{n_s} \tau_{f_s} \tag{4}$$

and it would also be greater than or equal to zero. We define a novel pruning estimator γ as

$$\gamma = \frac{\tau_{\mathcal{F}}}{n_s} + \beta \tag{5}$$

If the relative filter indices of the layer $l^{[k]}$ is denoted by a binary vector $V = [v1, v2, v3, \ldots, v_{n_s}]$ where each value of $V \in (0, 1)$. The relative filters index in V can be set to zero if the $|\tau|$ of the respective filter is less than γ otherwise to one as shown below.

$$V_{n_s} = \begin{cases} 0 & \text{if} \quad |\tau_i| < \gamma \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

We call γ as the pruning estimator, as it determines the number of filters that can be pruned from the each convolutional layer. Further, β was introduced to reduce or increase the set of the filters to be pruned. The earlier studies [18, 23, 35] have shown that the small magnitude weights or filters with smaller $\ell 1$-norm generates weaker activation maps. Therefore, the impact of the weaker activation maps on the final output of the model is less. In this context, the filters with smaller $\ell 1$-norm becomes pruning candidate compared to the filters with larger $\ell 1$-norm [23]. However, finding the optimal number of filters that can be pruned from the model is a major bottleneck. Therefore, the proposed pruning estimator (γ) eliminates the need of defining the manual pruning threshold for each layer.

In Eq. (5), β is pruning controller $\beta \in \mathbb{R}$ and its value varies between -2 and $+2$. The range of the pruning controller (β) was determined experimentally. β can be used for situations where the model needs to be prune more or less aggressively. It could be set to zero when we want to prune filters solely based on the mean absolute sum information. Otherwise, β could be set to some positive value if we want to prune more aggressively or to some negative value if we want to reduce the pruning of filters. We empirically found that pruning filters based on γ substantially perform better than other methods in which the pruning rate is given manual. To show that the proposed pruning estimator γ works for different models and to find the best trade-off between accuracy, % parameters pruned, and % FLOPs reduction, an extended analysis was made by performing regression statistical test and plotting β. The results have been added to Sect. 4.3.

2.2 Prune unimportant filters

In CNNs, a set of filters $F^{(h\times w\times c\times n)}$ are applied at each convolutional layer. Where, h and w is the height and width of the filter, c is the number of channels and, n is the number of filters applied in any convolutional layer $l \in L$. Each filter $f^{(h\times w\times c)} \in F$ produces one feature map, when all the feature maps are stacked together, they produce a feature map of size $((\frac{(d-h+2p)}{s}+1)\times(\frac{(d-h+2p)}{s}+1)\times n)$, and it becomes the input for the next $l+1$ layer. Where, p and s is the padding and stride, respectively, and d is the dimension of the input. The activation produced by $l-1$ convolutional layer is given by $(h^{[l-1]} \times w^{[l-1]} \times n^{[l-1]}$. Where $h^{[l-1]}$, $w^{[l-1]}$ is the dimension of the output and $n^{[l-1]}$ is the depth. This activation will work as an input for the layer l. Suppose, in any layer l, the filters of size $(h^{[l]} \times w^{[l]} \times n^{[l-1]})$ are applied. After applying convolutional, and nonlinearity, it will produce an activation/feature map of size $(h \times w \times N)$, N is the number of filters applied. Where, h and w are given by $((h^{[l-1]} - h^{[l]} + 2p)/s + 1)$. Figure 4 shows pruning of the filters from any layer l. In Fig. 4, left part shows the original un-pruned architecture. It should be noted that currently it produces activation for the $l+1$ layer as $(h \times w \times N)$. If at any point in time, it is found that the two filters out of the four are not contributing towards improving the model performance in layer l, then these two filters will be selected for pruning. The right part of the diagram shows that the two unimportant filters and their corresponding feature maps are pruned. Pruning of these filters and their corresponding feature maps will result in reducing the input for the $l+1$ layer. Now, as it is visible from Fig. 4 (right part) that after pruning of filters and their feature maps, the input for the next $l+1$ layer would be $(h \times w \times (N-2))$, before pruning it was $(h \times w \times N)$. Before pruning, the dimension of the weight tensor W was (4, $n^{[l-1]}$, $h^{[l]}$, $w^{[l]}$), after pruning, the new dimension of the W would be (2, $n^{[l-1]}$, $h^{[l]}$, $w^{[l]}$). $W[0]$ represent the weights of the first filters, $W[1]$ second filter, and so on.

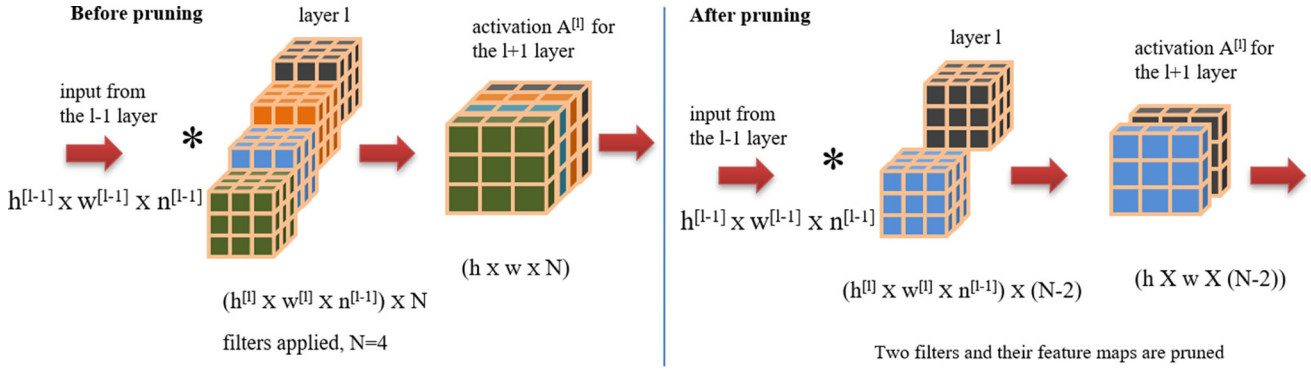

Fig. 4 Visualizing filter pruning (the left part of the diagram is the original, and right part shows the impact of pruning filters and their corresponding feature maps), * shows the convolutional operation between input and filters

To prune a set of filters from any layer l, all the filters are ranked according to their absolute sum in the proposed method. Then, γ is calculated for the layer. According to our approach, filters whose absolute sum is below the value of γ is a pruning candidate. We use a binary mask to mark whether any filter will be pruned or not from the layer. If the absolute sum of the filter f_i is less than γ, then the corresponding filter index is set to zero, otherwise one. A value of zero indicates that the filter is less important and is a pruning candidate. After getting the information of all the filters to be pruned, we performed one-shot pruning of all the filters from the layer. This process is repeated for all the convolutional layers. Whenever any filter is pruned from the layer l, its corresponding feature map is also pruned. Pruning of the feature map reduces the number of input channels in the next layer, so the numbers of channels are also reduced in the $l+1$ layer.

Once all the unimportant filters are pruned from all the layers, the new model is created with the remaining set of filters. The weights of the remaining filters are copied from the original architecture to a newly created architecture layer-wise. The pruning of the last convolutional layer affects the first dense layer. After pruning the last convolutional layer, the input connections in the first dense layer are changed, and weights are copied accordingly.

2.3 Fine-tuning

After pruning, the original model's performance degrades, and to compensate for the performance loss the pruned model needs to be fine-tuned. For this purpose, we took the pruned model and performed fine-tuning for more number of epochs on the original dataset. We conducted a number of experiments, and the experimental results support that the pruned model can recover the performance after fine-tuning. It is also worth noting that, since our proposed method is based on pruning of filters and does not make the resulting pruned architecture unstructured, the pruned model can directly be deployed on the target device without requiring special hardware or software [39]. Our method is also orthogonal to other compression and acceleration methods. The pruned model can further be processed by other methods such as low-rank factorization, and weight quantization to further compress and accelerate the model performance. In the next section, we have provided the experiments detail by applying the proposed method to popular convolutional architectures and datasets.

3 Experimental setup

In this section, we elucidate the details of the various experiments performed to evaluate the proposed method's effectiveness. The proposed method is evaluated on a variety of network architecture and datasets. All the experiments are performed on NVIDIA DGX V100 supercomputer with 128 GB RAM, 40,960 CUDA cores, 5120 tensor cores, and 960 TFLOPS speed. PyTorch [41] deep learning framework is used for implementing all the experiments.

3.1 Models and dataset

During experiments, we selected a fair amount of small and large DNN architectures such as AlexNet [31], VGG16 [43], and deeper network ResNet34 [20]. We performed experiments with CIFAR10 [32], CIFAR100 [32], and ImageNet [11] datasets. For CIFAR10 and CIFAR100 datasets, we used a modified version of the VGG16 model, and for the ImageNet dataset, we use the original VGG16 model. Standard dataset splits are used while training and testing the models on various datasets. Table 1 summarizes the details of models used in the experiments with their learnable parameters and FLOPs needed for a single image during inference. Table 1 also summarize the details of the CIFAR10, CIFAR100, and ImageNet datasets such as the

Table 1 Details of the convolutional architecture and datasets used in the experiments

Exp.	Model	Dataset	#Para	Classes	Image size	Train images	Test images	FLOPs
1	VGG16	CIFAR10	14.98M	10	$32 \times 32 \times 3$	50K	10K	313M
2	VGG16	CIFAR100	15M	100	$32 \times 32 \times 3$	50K	10K	314M
3	AlexNet	ImageNet	61.1M	1000	$224 \times 224 \times 3$	1.2M	50K	1.43B
4	VGG16	ImageNet	138.3M	1000	$224 \times 224 \times 3$	1.2M	50K	30.97B
5	ResNet34	ImageNet	21.8M	1000	$224 \times 224 \times 3$	1.2M	50K	3.68B

number of classes, training and test images, and the dimension of the images.

3.2 Experiment procedure and evaluation criteria

In general, we apply the following steps to achieve our research objectives discussed in Sect. 1.

- Train the model from scratch (CIFAR10 and CIFAR100 dataset) or use pre-trained model (ImageNet dataset).
- Apply the proposed filter pruning method (Algorithm 1) discussed in Sect. 2 to identify and prune unimportant filters from the model.
- Fine-tune the pruned model on the respective dataset to compensate for the accuracy loss due to the filter's pruning.

The performance of the original and pruned model is measured based on their top1 test accuracy (Eq. 7), % reduction in the FLOPs (Eq. 8), and % learnable parameters pruned (Eq. 9). The three evaluation criteria are given as

$$\text{acc} = \frac{\text{classification}^{\text{correct}}}{\#\text{images}} \times 100 \tag{7}$$

$$\text{flops} = \frac{(\text{FLOPs}^{\text{orig}} - \mathit{FLOPs}^{\text{pruned}})}{\text{FLOPs}^{\text{orig}}} \times 100 \tag{8}$$

$$\text{para} = \frac{(\text{Para}^{\text{orig}} - \text{Para}^{\text{pruned}})}{\text{Para}^{\text{orig}}} \times 100 \tag{9}$$

$\text{classification}^{\text{correct}}$ is the number of images correctly classified by the model, #images is the total number of images in the test set. $\text{FLOPs}^{\text{orig}}$ is the total number of FLOPs in the original model, $\text{FLOPs}^{\text{pruned}}$ is the total number of FLOPs in the pruned model. Similarly, $\text{Para}^{\text{orig}}$ and $\text{Para}^{\text{pruned}}$ is the number of learnable parameters in the original and pruned model, respectively.

3.3 Experiments on the CIFAR10 and CIFAR100 dataset

VGG16 model has 13 convolutional layers, and it has been used in various classification and detection problems. We used a variant of the VGG16 model with batch normalization [29] and without dropout [45] for CIFAR10 and CIFAR100 experiments. The modified architecture has two dense layers, one with 512 nodes that receive input directly from the last convolutional layer and one output layer. We modified the number of nodes in the last dense layer to 10 and 100 to match with the CIFAR10 and CIFAR100 datasets, respectively. The base architecture is trained from scratch to get top1 accuracy of 92.25% and 69.37%, respectively, for CIFAR10 and CIFAR100 experiments. The base model is trained for 160 epochs with an SGD optimizer and 0.9 momentum. The learning rate was set to 0.1, and the weight decay of 1e−4 is applied. The learning rate is decayed by ten after every 30 epochs. For the CIFAR100 experiment, all the training parameters were the same as used while working with CIFAR10. The pruned model is fine-tuned for 80 epochs with a 0.01 learning rate.

3.4 Experiments on the ImageNet dataset

To test the method's generalization ability and effectiveness, we further evaluate the proposed method on a large-scale ImageNet dataset. The method is evaluated on three convolutional architectures AlexNet, VGG16, and ResNet34, and presented a comparative study for VGG16 and ResNet34 in the next section. For ImageNet experiments, we use the pre-trained models from PyTorch [41] deep learning framework as base architectures. The pruned models are fine-tuned with an SGD optimizer with a momentum of 0.9. The batch size was 128, the learning rate was set to 0.001, and the weight decay of 1e−4 is applied. Images are resized to 256×256 and randomly center cropped at 224×224. Random horizontal flip is used as data augmentation during training, and in the testing center crop is used. No other data augmentation techniques are used.

The initial experiments are performed with the AlexNet model. The pre-trained PyTorch AlexNet [31] model has 61.1M parameters and has 1.43B FLOPs without batch normalization layers. The top1 and top5 accuracy of the original pre-trained model are 56.45% and 79.09%, respectively. For VGG16 experiment, we use the original model with 13 convolutional and three dense layers without batch normalization from PyTorch. VGG16 has 138.3M learnable parameters and 30.97B FLOPs for a single inference. For the sake of comparison, we consider multiplication and addition as two separate FLOPs for VGG16 model.

The original VGG16 model is tested on the 50K ImageNet validation set and recorded 71.26% and 90.22% top1 and top5 accuracy, respectively. The pruned model is fine-tuned for 75 epochs. VGG16 is a simple network architecture in which the layers are stacked on top of each other. Next, we evaluate our method on an even more complex network. One such network is the ResNet34 [20] which is one of the ResNet network variants. ResNet34 is a 34-layer network with skip connections. It has four stages of residual blocks and uses projection shortcuts during the down-sampling of the feature maps. We test the original model on the ImageNet 50K validation set and recorded 72.04% top1 accuracy. The pruned model is fine-tuned for 50 epochs.

4 Results and discussion

4.1 CIFAR10 and CIFAR100 experiment

Table 2 shows the VGG16 pruned model details and its comparison with state-of-the-art methods on the CIFAR10 dataset. It is worth noting from Table 2 that the proposed method pruned more than 80% parameters and reduces 71.82% FLOPs with marginal 0.04% drop in the top1 accuracy. Another observation is that the best performing method [26] reduces FLOPs by 64.5%; however, the accuracy loss is more than ours. Also, the original VGG16 model has 4224 filters, whereas the pruned model has only 1887 filters. In Table 2, [38] count multiplication and addition as two separate FLOPs. We count multiplication and addition as 1 FLOP (MAC) operation.

Figure 5 shows the number of filters before and after pruning the model for the CIFAR10 and CIFAR100 dataset. It can be seen from Fig. 5 that all the convolutional layers contain unimportant weights filters, and a substantial number of filters are pruned from all the layers. Figure 5 also depicts layers with more filters containing more candidate filters for pruning. From the last six layers, more than 50% filters are pruned without significantly hurting the model's performance.

The percentage of FLOPs pruned for CIFAR10 dataset and VGG16 model is visualized in Fig. 6. The reduction in FLOPs is higher for the layers from 8 to 12, more than 80% FLOPs are reduced in these layers. The last layer is directly connected to the dense layer; still, 67% FLOPs are reduced in this layer. It is also important to note that the pruned model is able to get competitive accuracy with $\beta = 0$, pruning filters are determined only based on the mean absolute sum.

Table 3 shows the VGG16-CIFAR100 pruned model details and its comparison with other state-of-the-art methods. It is worth noting from Table 3 that the proposed method is able to prune more than 75% parameters and reduce 72.83% FLOPs with 1.79% drop in the top1 accuracy. Further improvement in the accuracy could be achieved by testing β for negative values. The method performs better than [38], and [54] in terms of the number of parameters pruned, and FLOP reduction. Moreover, a total of 2045 out of 4224 filters are pruned from various convolutional layers. Like the CIFAR10 experiment, the CIFAR100 experiment also shows that VGG16 contains many unimportant filters, and more than 50% filters can be pruned from the model. The percentage of FLOPs pruned for CIFAR10 and CIFAR100 dataset with the VGG16 model is visualized in Fig. 6.

The CIFAR10 and CIFAR100 dataset experiments show that many filters can be pruned without significantly affecting the model performance. The pruned model will have to perform fewer FLOPs during inference, and it directly improves the inference performance of the model in terms of latency, computational and battery power

Table 2 Comparison of the VGG16-CIFAR10 pruned model with other state-of-the-art methods, $\beta = 0$

Method	Top1 acc (±)	#Para left	%Para pruned	#FLOPs left	%FLOPs reduction
[38]	0.14	2.30M	88.5	3.91×10^8	51
[35]	0.15	5.4M	64	2.06×10^8	34
[54]	− 0.07	3.92M	73.34	1.9×10^8	39.1
[26]	− 1.9	–	–	–	64.5
Ours	− 0.04	2.86M	80.91	8.8×10^7	71.82

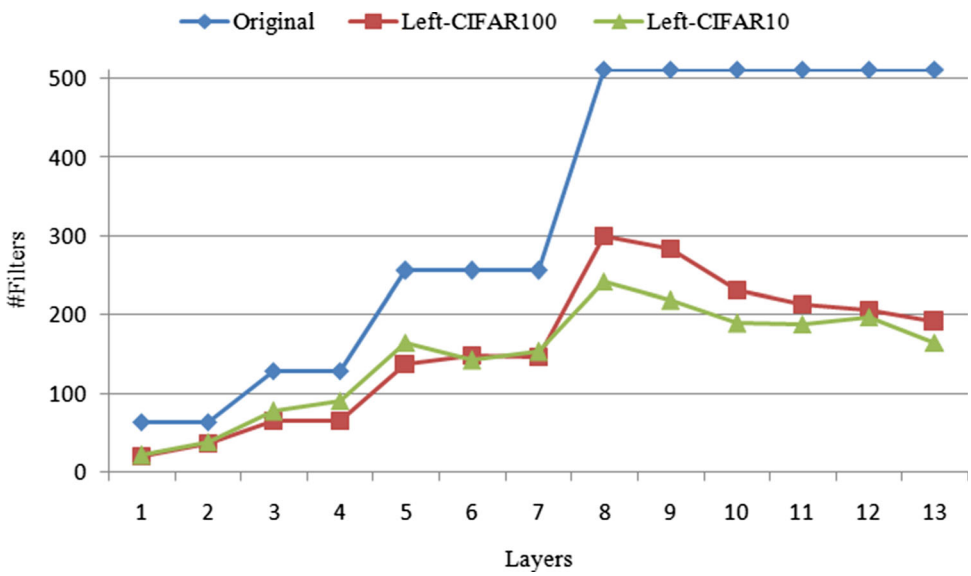

Fig. 5 Number of filters left in the convolutional layers before and after pruning (VGG16 on CIFAR10 and CIFAR100 dataset)

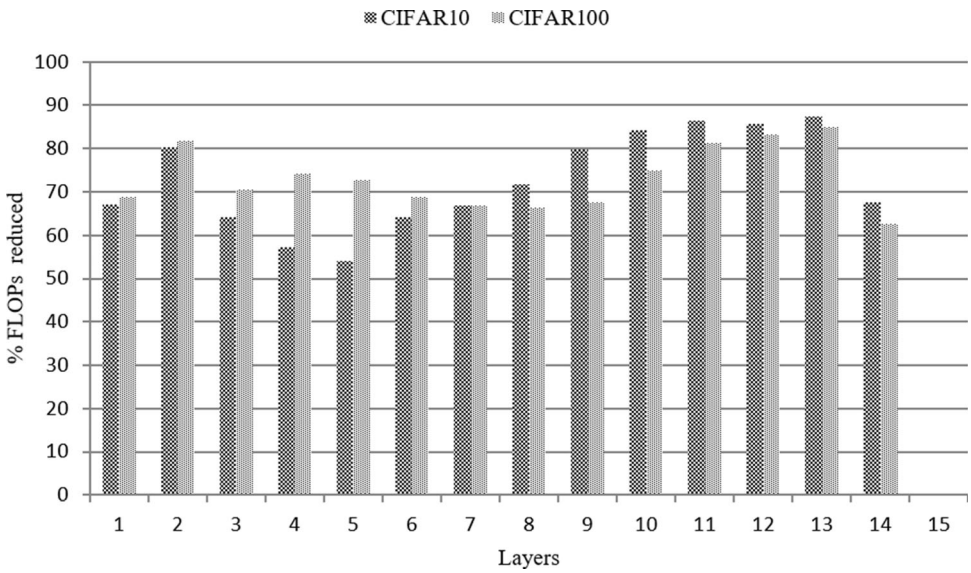

Fig. 6 Layer-wise percentage of FLOPs reduction for VGG16 on CIFAR10 and CIFAR100 dataset

Table 3 Comparison of the VGG16-CIFAR100 pruned model with other state-of-the-art methods, $\beta = 0$

Method	Top1 acc (±)	#Para left	%Para pruned	#FLOPs left	%FLOPs reduction
[38]	0.22	5M	75.1	5.01×10^8	37.1
[54]	0.07	9.14M	37.87	2.56×10^8	18.05
Ours	− 1.79	3.6M	75.86	8.5×10^7	72.83

required. CIFAR10 and CIFAR100 experiments confirm and relate with our first research objective that the inference performance of the DNN can be improved by reducing the time-consuming FLOPs.

4.2 ImageNet experiments

Table 4 shows the detail of the FLOPs of the convolutional and dense layer of the original and pruned AlexNet model. It can be seen from Table 4, 91.81% of the total FLOPs come from computationally intensive convolutional layers, and the contribution of the dense layer is only 8.19%. After

Table 4 The number of FLOPs of AlexNet-ImageNet model before and after pruning ($\beta = -3$)

Original model		Pruned model	
Layers	#FLOPs	#FLOPs left	%FLOPs reduction
Conv1	1E+08	7.5E+07	46.87
Conv2	4E+08	1.4E+08	68.42
Conv3	2E+08	9.8E+07	56.22
Conv4	3E+08	1.5E+08	49.32
Conv5	2E+08	1.2E+08	38.50
FC1	8E+07	6.8E+07	10.54
FC2	3E+07	3E+07	0
FC3	8E+06	8E+06	0

fine-tuning the model with $\beta = 0$, the top1 accuracy drop was 6%, and with $\beta = -3$, the model is able to reduce the accuracy drop to 1.95%. AlexNet is a comparatively small model and cannot gain accuracy if pruned more aggressively. However, our method pruned 316 filters out of 1152 and reduce the FLOPs from 1.43B to 0.69B. The results of the AlexNet further motivated us to conduct experiments on an even deeper model; VGG16 and ResNet34.

Table 5 shows the number of FLOPs of the VGG16 model before and after pruning, along with the percentage of parameters pruned away, as shown in the last column. It is important to note that, with the proposed method, the number of FLOPs is reduced from 30.97B to 6.97B, pruning 77.47% of the total and bringing 4.45x acceleration in the model performance. It also resulted in lowering 40.42% parameters of the model. The last convolutional layer filters are reduced from 512 to 289, resulting in 44% parameter saving and 43.45% reductions in the FLOPs in the first dense layer. More than 75% of the FLOPs are reduced in the last six convolutional layers. It implies that these layers contain more than 50% lesser importance filters, and can be pruned from the layers 8, 9, 10, and 11.

Table 6 Comparison of the VGG16-ImageNet pruned model with other methods, $\beta = 0$

Method	FLOPs left	Top1 acc (±)
[56]	6.2B	− 3.3
[40]	9.34B	− 3.81
Ours	6.97B	− 2.41

The comparison of the proposed method with other state-of-the-art methods is given in Table 6. It can be seen from Table 6 that the proposed method performed better than [56], which causes − 3.3% accuracy drop, while in our case the accuracy drop is − 2.41%. The authors did not provide the number of FLOPs reduced; instead, it is reported that 5x reduction is achieved. We calculated the FLOPs based on the reported 5x reduction out of the total VGG16 FLOPs. Our method also performs superior to [40]. In [40], the number of FLOPs reduced to 9.34B with − 3.81% drop in the top1 accuracy. It is worth noting that, for the VGG16-ImageNet experiment, the β was set to 0,

Table 5 Details of the VGG16 original and pruned model with ImageNet dataset, $\beta = 0$

Original model			Pruned model			
Layers	FLOPs	Filters	FLOPs	Filters	%FLOP reduction	%Para pruned
Conv1	1.8E+08	64	7.9E+07	28	56.25	56
Conv2	3.7E+09	64	7.6E+08	30	79.42	79
Conv3	1.8E+09	128	3.9E+08	58	78.72	79
Conv4	3.7E+09	128	6.3E+08	48	82.98	83
Conv5	1.9E+09	256	3.5E+08	129	81.08	83
Conv6	3.7E+09	256	9.7E+08	133	73.81	74
Conv7	3.7E+09	256	8.6E+08	115	76.65	77
Conv8	1.9E+09	512	8.2E+08	243	78.67	79
Conv9	3.7E+09	512	8.2E+08	240	77.75	78
Conv10	3.7E+09	512	8.3E+08	245	77.56	78
Conv11	9.3E+08	512	2.2E+08	250	76.62	77
Conv12	9.3E+08	512	2.3E+08	265	74.72	75
Conv13	9.3E+08	512	2.7E+08	289	70.48	71
FC1	2.1E+08	–	1.2E+08	–	43.55	44
FC2	3.4E+07	–	3.4E+07	–	0	0
FC3	8E+06	–	8E+06	–	0	0
Total	30.97B	4224	6.97B	2073	77.47	40.42

Table 7 Comparison of the ResNet34 pruned model with other methods

Method	Top1 acc (±)	%Para pruned	%FLOPs reduction
[35]—A	− 0.67	7.69	15.50
[35]—B	− 1.06	10.80	24.20
[1]	− 0.31	26.53	28.12
Yu et al. [51]	− 0.28	27.14	27.32
Ours, $\beta = 0$	1.37	13.01	25.73
Ours, $\beta = 1$	0.83	17.32	35.18
Ours, $\beta = 2$	0.37	20.16	41.94

further improvement in accuracy can be achieved with negative values of β.

Table 7 shows the results of the ResNet34 pruned model and comparison with other methods. It can be seen from Table 7, our method outperformed [35] with $\beta = 0$. To further get the competitive performance we set $\beta = 1$, and $\beta = 2$ as shown in Table 7. It is worth noting that, with $\beta = 1$ our method outperformed other state-of-the-art methods and achieved 0.83% improvements in the top1 accuracy. Similarly, for $\beta = 2$, the method is able to reduce 41.94% FLOPs and achieved 0.37% improvement in the top1 accuracy.

4.3 Extended analysis

The regression statistical test was performed between the varied values of the pruning controller β and the %parameters pruned, %FLOP reduction, and accuracy to show the significance of the proposed pruning estimator γ. Here, β was taken as the independent variable and other parameters were taken as dependent variables. These tests were performed on the ResNet34 and VGG16 pruned model on the ImageNet dataset. The regression statistical test finds the relationship between the independent and the dependent variables. In this regard, the null and the alternate hypotheses were established. The null hypothesis (H_0 : the pruning estimator γ does not finds an optimal trade-off between the accuracy, %parameters pruned, and %FLOPs reduction. The alternate hypothesis H_1: the pruning estimator γ does finds an optimal trade-off between the accuracy, %parameters pruned, and %FLOPs reduction. The significance level α was set to 0.05. The p-value below 0.05 signifies that there is sufficient evidence to reject the null hypothesis.

Different experiments were performed on the ResNet34 and VGG16 model with varied values of β (− 2 to + 2). In each experiment, the original models were pruned according to the pruning estimator γ and the pruned models were fine-tuned on the ImageNet dataset. For the ResNet34 model, with $\alpha = 0.05$, the p-value for β and %parameters pruned, %FLOP reduction, and accuracy was 0.000135, 0.000173, and 0.002089, respectively. The smaller p values indicate that there is sufficient evidence to reject the null hypothesis. For the VGG16 model, with $\alpha = 0.05$, the p value for β and %parameters pruned, %FLOP reduction, and accuracy was <0, 0.001178, and 0.003983, respectively. Here, also the smaller p values supports that there is a sufficient evidence to reject the null hypothesis. For β less than zero, the %parameters pruned, %FLOPs reduction were less and the accuracy loss was also less. For β greater than zero, the %parameters pruned, %FLOPs reduction were more and the accuracy loss was also more. From this, it can be concluded that the optimal values of the β for both the models is close to zero, and it finds an optimal trade-off between the accuracy and the %parameters pruned and the %FLOPs reduction.

To further show that the proposed pruning estimator γ finds an optimal trade-off between reduction (parameters, FLOPs) and the accuracy, we plotted the value of β and the

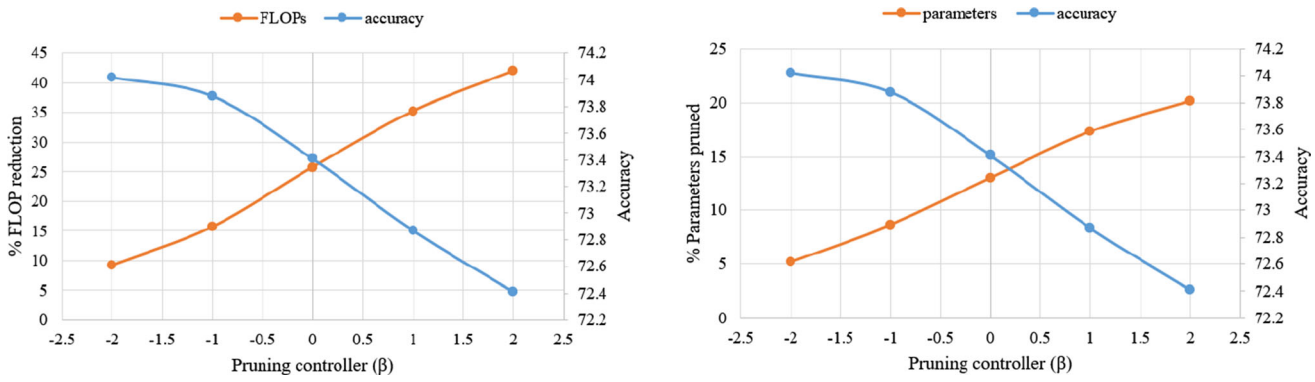

Fig. 7 ResNet34-ImageNet experiment: The left diagram shows the the plot of β versus %FLOP reduction and accuracy. The right diagram shows the plot of β versus %parameters pruned and accuracy

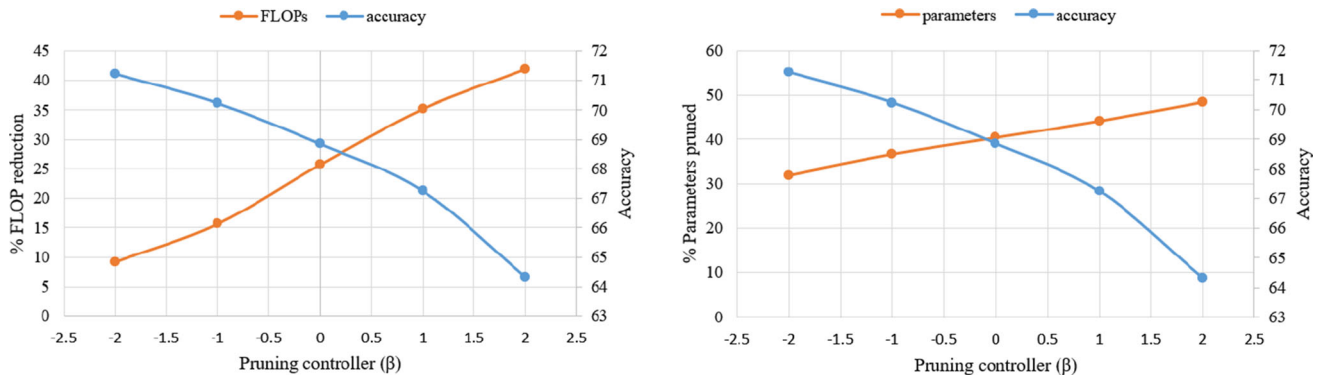

Fig. 8 VGG16-ImageNet experiment: The left diagram shows the plot of β versus %FLOP reduction and accuracy. The right diagram shows the plot of β versus %parameters pruned and accuracy

%parameters pruned, %FLOPs reduction, and accuracy for the ImageNet experiments. Figure 7 shows the graph for the ResNet34 model. It can be seen from Fig. 7 that the optimal trade-off is achieved when β is close to zero in terms of the %parameters pruned, the %FLOPs reduction, and the accuracy. Similarly, Fig. 8 shows the graph for the VGG16 model. It can also be seen from Fig. 8 that the optimal trade-off exists when β is close to zero in terms of the %parameters pruned, the %FLOPs reduction, and the accuracy.

4.4 Discussion

There is no doubt that filter pruning is a popular way to accelerate convolutional architecture's inference performance. However, manual pruning methods require a lot of tuning to find out the correct pruning rate for each layer. In the proposed method, the selection of filters for pruning does not require any manual tuning. Besides, our approach has a single parameter that can be further tuned to control the pruning process if needed. Figure 9 shows the comparison of the original and pruned models in terms of the % of FLOPs reduced from different models on different datasets. It should be noted from Fig. 9, the proposed method is significantly able to reduce FLOPs (research objective 1), and the proposed method is also generalized across various network architectures (research objective 2). ImageNet trained VGG16, and ResNet34 have been used for real-time object detection and classification applications. The pruned model can also be used as a base classifier in object detection tasks to improve the object detection model's inference performance for real-time applications. ResNet34 is an efficient model than VGG16 in terms of the number of parameters and FLOPs. However, it is worth noting that the pruned ResNet34 has 41.94% fewer FLOPs than the original model with 0.37%

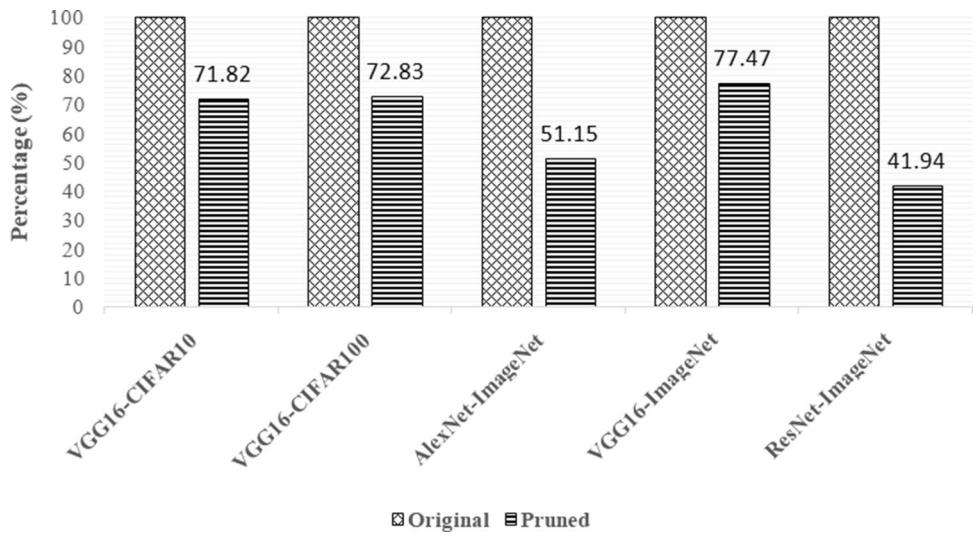

Fig. 9 Percentage of FLOPs reduction in different model and datasets

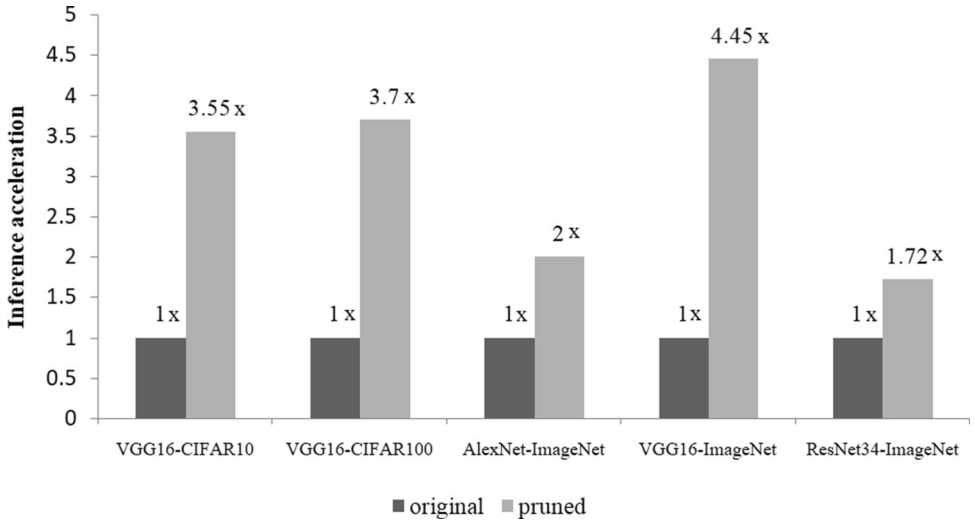

Fig. 10 Inference performance comparison of the original and pruned models based on the number of FLOPs reduction

accuracy gain. The ImageNet experiments on the VGG16, ResNet34 and their improved performance compared to other methods show that the proposed method effectively reduces the inference-cost of the deep learning models.

Our other observation is that depending upon the application's requirement or the target device, the proposed method can be customized. For situations where a little accuracy loss can be compromised, our method can be used to prune the model more aggressively. Simultaneously, for applications where accuracy is more critical, our method can be customized to maintain the accuracy loss within an acceptable range. Figure 10 shows the overall improvement in the model's inference performance by comparing the original and pruned model FLOPs. The initial performance of the un-pruned model is shown as 1x. It can be seen from Fig. 10, the performance of the pruned model has increased by 3.55x and 3.7x, respectively, for CIFAR10 and CIFAR100 dataset. Also, on the large-scale ImageNet dataset, the performance of the VGG16, and ResNet34 is improved by 4.45x and 1.72x, respectively.

5 Conclusion

Nowadays, DNNs are getting deeper, requiring high computational and battery power during inference. Pruning has evolved as an essential method to remove the model's unimportant parameters while least hurting the performance. In some cases, it also improves the performance of the model. In this paper, we proposed a heuristic-based novel automatic filter selection and pruning method to accelerate the convolutional architecture's inference performance. The experimental results suggest that the proposed method significantly improves the inference performance for real-time applications and devices with limited resources. On the popular ResNet34 architecture trained with the ImageNet dataset, our method is able to reduce FLOPs by 41.94% with 0.37% improvements in the top1 accuracy. One of our method's main advantages is that the model pruned with the proposed method can further be compressed and accelerated by applying other compression and acceleration techniques. The experiments on various model architecture and datasets show the superiority of the proposed method over state-of-the-art methods. Future work can include exploiting the similarity between the filters in a trained model because similar filters extract similar features and create redundancy in the feature maps.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Ayinde BO, Inanc T, Zurada JM (2019) Redundant feature pruning for accelerated inference in deep neural networks. Neural Netw 118:148–158
2. Badrinarayanan V, Kendall A, Cipolla R (2017) Segnet: a deep convolutional encoder–decoder architecture for image segmentation. IEEE Trans Pattern Anal Mach Intell 39(12):2481–2495
3. Cai Z, He X, Sun J, Vasconcelos N (2017) Deep learning with low precision by half-wave gaussian quantization. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5918–5926
4. Chen G, Choi W, Yu X, Han T, Chandraker M (2017) Learning efficient object detection models with knowledge distillation. In: Advances in neural information processing systems, pp 742–751

5. Chen S, Zhao Q (2018) Shallowing deep networks: layer-wise pruning based on feature representations. IEEE Trans Pattern Anal Mach Intell
6. Cheng J, Wu J, Leng C, Wang Y, Hu Q (2017) Quantized CNN: a unified approach to accelerate and compress convolutional networks. IEEE Trans Neural Netw Learn Syst
7. Cheng Y, Wang D, Zhou P, Zhang T (2017b) A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282
8. Cheng Y, Wang D, Zhou P, Zhang T (2018) Model compression and acceleration for deep neural networks: the principles, progress, and challenges. IEEE Signal Process Mag 35(1):126–136
9. Choudhary T, Mishra V, Goswami A, Sarangapani J (2020) A comprehensive survey on model compression and acceleration. Artif Intell Rev 53:5113–5155
10. Courbariaux M, Bengio Y, David JP (2015) Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems, pp 3123–3131
11. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) Imagenet: A large-scale hierarchical image database. In: IEEE conference on computer vision and pattern recognition. IEEE, pp 248–255
12. Denil M, Shakibi B, Dinh L, De Freitas N, et al (2013) Predicting parameters in deep learning. In: Advances in neural information processing systems, pp 2148–2156
13. Girshick R (2015) Faster r-CNN: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, pp 91–99
14. Gong Y, Liu L, Yang M, Bourdev L (2015) Compressing deep convolutional networks using vector quantization. Under review as a conference paper at ICLR
15. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680
16. Graves A, Ar Mohamed, Hinton G (2013) Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 6645–6649
17. Guo Y, Yao A, Chen Y (2016) Dynamic network surgery for efficient DNNs. In: Advances In neural information processing systems, pp 1379–1387
18. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems, pp 1135–1143
19. Han S, Mao H, Dally WJ (2016) Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. Published as a conference paper at ICLR
20. 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
21. 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
22. 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
23. He Y, Dong X, Kang G, Fu Y, Yan C, Yang Y (2019) Asymptotic soft filter pruning for deep convolutional neural networks. IEEE Trans Cybern 50(8):3594–3604
24. Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531
25. Horowitz M (2014) 1.1 computing's energy problem (and what we can do about it). In: 2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC). IEEE, pp 10–14
26. 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
27. Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2016) Binarized neural networks. In: Advances in neural information processing systems, pp 4107–4115
28. Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2017) Quantized neural networks: training neural networks with low precision weights and activations. J Mach Learn Res 18(1):6869–6898
29. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning, pp 448–456
30. Jaderberg M, Vedaldi A, Zisserman A (2014). Speeding up convolutional neural networks with low rank expansions. In: Proceedings of the British machine vision conference. BMVA Press
31. Krizhevsky A (2014) One weird trick for parallelizing convolutional neural networks. arXiv preprint arXiv:1404.5997
32. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. Tech. rep, Citeseer
33. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp 1097–1105
34. LeCun Y, Denker JS, Solla SA (1990) Optimal brain damage. In: Advances in neural information processing systems, pp 598–605
35. Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2017) Pruning filters for efficient convnets. Published as a conference paper at ICLR
36. Liu S, Lin Y, Zhou Z, Nan K, Liu H, Du J (2018). On-demand deep model compression for mobile devices: a usage-driven model selection framework. In: Proceedings of the 16th annual international conference on mobile systems, applications, and services, pp 389–400
37. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC (2016) Ssd: Single shot multibox detector. In: European conference on computer vision. Springer, pp 21–37
38. 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
39. Liu Z, Sun M, Zhou T, Huang G, Darrell T (2019) Rethinking the value of network pruning. Published as a conference paper at ICLR
40. Luo JH, Zhang H, Zhou HY, Xie CW, Wu J, Lin W (2018) Thinet: Pruning CNN filters for a thinner net. IEEE Trans Pattern Anal Mach Intell https://doi.org/10.3390/electronics9081209
41. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in pytorch
42. Ren S, He K, Girshick R, Sun J (2015) Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, pp 91–99
43. Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. Published as a conference paper at ICLR
44. Srinivas S, Babu RV (2015) Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149
45. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958

46. Swaminathan S, Garg D, Kannan R, Andres F (2020) Sparse low rank factorization for deep neural network compression. Neurocomputing. https://doi.org/10.1016/j.neucom.2020.02.035
47. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9
48. Tung F, Mori G (2018) Deep neural network compression by in-parallel pruning-quantization. IEEE Trans Pattern Anal Mach Intell. https://doi.org/10.1109/TPAMI.2018.2886192
49. Wu X, Wu Y, Z Y (2016) Binarized neural networks on the imagenet classification task. arXiv preprint arXiv:1604.03058
50. Yang TJ, Chen YH, Sze V (2017) Designing energy-efficient convolutional neural networks using energy-aware pruning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5687–5695
51. Yu R, Li A, Chen CF, Lai JH, Morariu VI, Han X, Gao M, Lin CY, Davis LS (2018) Nisp: Pruning networks using neuron importance score propagation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 9194–9203
52. Zhang C, Bengio S, Hardt M, Recht B, Vinyals O (2016a) Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530
53. Zhang X, Zou J, He K, Sun J (2016) Accelerating very deep convolutional networks for classification and detection. IEEE Trans Pattern Anal Mach Intell 38(10):1943–1955
54. Zhao C, Ni B, Zhang J, Zhao Q, Zhang W, Tian Q (2019) Variational convolutional neural network pruning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2780–2789
55. Zhou A, Yao A, Guo Y, Xu L, Chen Y (2017) Incremental network quantization: towards lossless CNNs with low-precision weights. arXiv preprint arXiv:1702.03044
56. Zhou Y, Zhang Y, Wang Y, Tian Q (2019) Accelerate CNN via recursive bayesian pruning. In: Proceedings of the IEEE international conference on computer vision, pp 3306–3315
57. Zhu M, Gupta S (2017) To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878