To Filter Prune, or to Layer Prune, That Is The Question

Sara Elkerdawy[1][0000−0002−9607−3225], Mostafa Elhoushi[2], Abhineet Singh[1], Hong Zhang[1], and Nilanjan Ray[1]

[1] Department of Computing Science, University of Alberta, Canada
{elkerdaw, asingh1, hzhang, nray1}@ualberta.ca
[2] Toronto Heterogeneous Compilers Lab, Huawei, Canada

Abstract. Recent advances in pruning of neural networks have made it possible to remove a large number of filters or weights without any perceptible drop in accuracy. The number of parameters and that of FLOPs are usually the reported metrics to measure the quality of the pruned models. However, the gain in speed for these pruned models is often overlooked in the literature due to the complex nature of latency measurements. In this paper, we show the limitation of filter pruning methods in terms of latency reduction and propose LayerPrune framework. LayerPrune presents a set of layer pruning methods based on different criteria that achieve higher latency reduction than filter pruning methods on similar accuracy. The advantage of layer pruning over filter pruning in terms of latency reduction is a result of the fact that the former is not constrained by the original model's depth and thus allows for a larger range of latency reduction. For each filter pruning method we examined, we use the same filter importance criterion to calculate a per-layer importance score in one-shot. We then prune the least important layers and fine-tune the shallower model which obtains comparable or better accuracy than its filter-based pruning counterpart. This one-shot process allows to remove layers from single path networks like VGG before fine-tuning, unlike in iterative filter pruning, a minimum number of filters per layer is required to allow for data flow which constraint the search space. To the best of our knowledge, we are the first to examine the effect of pruning methods on latency metric instead of FLOPs for multiple networks, datasets and hardware targets. LayerPrune also outperforms handcrafted architectures such as Shufflenet, MobileNet, MNASNet and ResNet18 by 7.3%, 4.6%, 2.8% and 0.5% respectively on similar latency budget on ImageNet dataset. [1]

Keywords: CNN pruning, layer pruning, filter pruning, latency metric

1 Introduction

Convolutional Neural Networks (CNN) have become the state-of-the-art in various computer vision tasks, e.g., image classification [1], object detection [2],

[1] Code is available at https://github.com/selkerdawy/filter-vs-layer-pruning

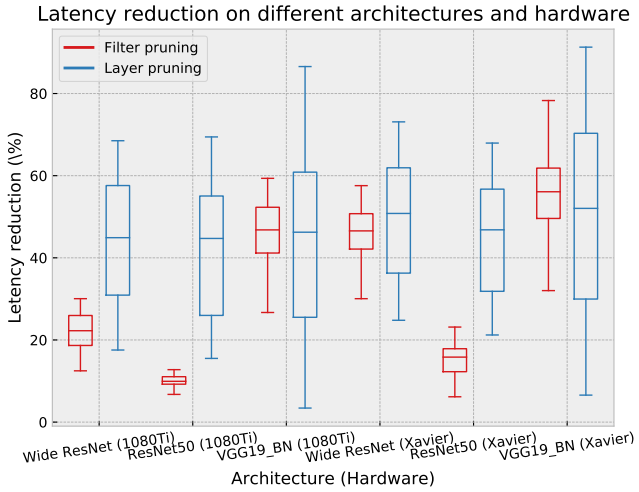

Fig. 1: Example of 100 randomly pruned models per boxplot generated from different architectures. The plot shows layer pruned models have a wider range of attainable latency reduction consistently across architectures and different hardware platforms (1080Ti and Xavier). Latency is estimated using 224x224 input image and batch size=1.

depth estimation [3]. These CNN models are designed with deeper [4] and wider [5] convolutional layers with a large number of parameters and convolutional operations. These architectures hinder deployment on low-power devices, e.g, phones, robots, wearable devices as well as real-time critical applications, such as autonomous driving. As a result, computationally efficient models are becoming increasingly important and multiple paradigms have been proposed to minimize the complexity of CNNs.

A straight forward direction is to manually design networks with a small footprint from the start such as [6,7,8,9,10]. This direction does not only require expert knowledge and multiple trials (e.g up to 1000 neural architectures explored manually [11]), but also does not benefit from available, pre-trained large models. Quantization [12,13] and distillation [14,15] are two other techniques, which utilize the pre-trained models to obtain smaller architectures. Quantization reduces bit-width of parameters and thus decreases memory footprint, but requires specialized hardware instructions to achieve latency reduction. While distillation trains a pre-defined smaller model (student) with guidance from a larger pre-trained model (teacher) [14]. Finally, model pruning aims to automatically remove the least important filters (or weights) to reduce the number of parameters or FLOPs (i.e indirect measures). However, prior work [16,17,18] showed that neither number of pruned parameters nor FLOPs reduction directly correlate with latency (i.e a direct measure) consumption. Latency reduction, in that case, depends on various aspects, such as the number of filters per layer

(signature) and the deployment device. Most GPU programming tools require careful compute kernels[2] tuning for different matrices shapes (e.g., convolution weights) [19,20]. These aspects introduce non-linearity in modeling latency with respect to the number of filters per layer. Recognizing the limitations in terms of latency or energy by simply pruning away filters, recent works [17,21,16] proposed optimizing directly over these direct measures. These methods require per hardware and architecture latency measurements collection to create lookup-tables or latency prediction models which can be time-intensive. In addition, these filter pruned methods are bounded by the model's depth and can only reach a limited goal for latency consumption.

In this work, we show the limitations of filter pruning methods in terms of latency reduction. Fig. 1 shows the range of attainable latency reduction on randomly generated models. Each box bar summarizes the latency reduction of 100 random models with filter and layer pruning on different network architectures and hardware platforms. For each filter pruned model i, a pruning ratio $p_{i,j}$ per layer j such that $0 \leq p(i,j) \leq 0.9$ is generated thus models differ in signature/width. For each layer pruned model, M layers out of total L layers (dependent on the network) are randomly selected for retention such that $1 \leq M \leq L$ thus models differ in depth. As to be expected, layer pruning has a higher upper bound in latency reduction compared to filter pruning especially on modern complex architectures with residual blocks. However, we want to highlight quantitatively in the plot the discrepancy of attainable latency reduction using both methods. Filter pruning is not only constrained by the depth of the model but also by the connection dependency in the architecture. An example of such connection dependency is the element-wise sum operation in the residual block between identity connection and residual connection. Filter pruning methods commonly prune in-between convolution layers in a residual to respect the number of channels and spatial dimensions. BAR [22] proposed an atypical residual block that allows mixed-connectivity between blocks to tackle the issue. However, this requires special implementations to leverage the speedup gain. Another limitation in filter pruning is the iterative process and thus is constrained to keep a minimum number of filters per layer during optimization to allow for data passing. LayerPrune performs a one-shot pruning before fine-tuning and thus it allows for layer removal even from single path networks.

Motivated by these points, what remains to ask is how well do layer pruned models perform in terms of accuracy compared to filter pruned methods. Fig. 2 shows accuracy and images per second between our LayerPrune, several state-of-the-art pruning methods, and handcrafted architectures. In general, pruning methods tend to find better quality models than handcrafted architectures. It is worth noting that filter pruning methods such as ThiNet [23] and Taylor [24] show small speedup gain as more filters are pruned compared to LayerPrune. That shows the limitation of filter pruning methods on latency reduction.

[2] A compute kernel refers to a function such as convolution operation that runs on a high throughput accelerator such as GPU

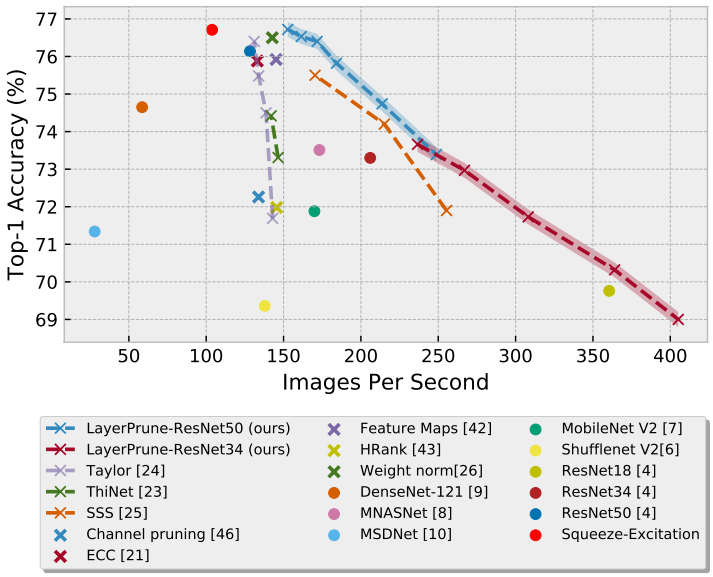

Fig. 2: Evaluation on ImageNet between our LayerPrune framework, handcrafted architectures (dots) and pruning methods on ResNet50 (crosses). Inference time is measured on 1080Ti GPU.

2 Related Work

We divide existing pruning methods into four categories: weight pruning, hardware-agnostic filter pruning, hardware-aware filter pruning and layer pruning.

Weight pruning. An early major category in pruning is individual weight pruning (unstructured pruning). Weight pruning methods leverage the fact that some weights have minimal effect on the task accuracy and thus can be zeroed-out. In [25], weights with small magnitude are removed and in [26], quantization is further applied to achieve more model compression. Another data-free pruning is [27] where neurons are removed iteratively from fully connected layers. L_0-regularization based method [28] is proposed to encourage network sparsity in training. Finally, in lottery ticket hypothesis [29], the authors propose a method of finding winning tickets which are subnetworks from random initialization that achieve higher accuracy than the dense model. The limitation of the unstructured weight pruning is that dedicated hardware and libraries [30] are needed to achieve speedup from the compression. Given our focus on latency and to keep the evaluation setup simple, we do not consider these methods in our evaluation.

Hardware-agnostic filter pruning. Methods in this category (also known as structured pruning) aim to reduce the footprint of a model by pruning filters without any knowledge of the inference resource consumption. Examples of these are [24,31,23,32,33], which focus on removing the least important filters and obtaining a slimmer model. Earlier filter-pruning methods [23,33] required layer-wise sensitivity analysis to generate the signature (i.e number of filters per layer) as a prior and remove filters based on a filter criterion. The sensitivity analysis is computationally expensive to conduct and becomes even less feasible for deeper models. Recent methods [24,31,32] learn a global importance measure removing the need for sensitivity analysis. Molchanov et al. [24] propose a Taylor approximation on the network's weights where the filter's gradients and norm are used to approximate its global importance score. Liu et al. [31] and Wen et al. [32] propose sparsity loss for training along with the classification's cross-entropy loss. Filters whose criterion are less than a threshold are removed and the pruned model is finally fine-tuned. Zhao et al. [34] introduce channel saliency that is parameterized as Gaussian distribution and optimized in the training process. After training, channels with small mean and variance are pruned. In general, methods with sparsity loss lack a simple approach to respect a resource consumption target and require hyperparameter tuning to balance different losses.

Hardware-aware filter pruning. To respect a resource consumption budget, recent works [35,17,21,36] have been proposed to take into consideration a resource target within the optimization process. NetAdapt [17] prunes a model to meet a target budget using a heuristic greedy search. A lookup table is built for latency prediction and then multiple candidates are generated at each pruning iteration by pruning a *ratio* of filters from each layer independently. The candidate with the highest accuracy is then selected and the process continues to the next pruning iteration with a progressively increasing *ratio*. On the other hand, AMC [36] and ECC [21] propose an end-to-end constrained pruning. AMC utilizes reinforcement learning to select a model's signature by trial and error. ECC simplifies the latency reduction model as a bilinear per-layer model. The training utilizes the alternating direction method of multipliers (ADMM) algorithm to perform constrained optimization by alternating between network weight optimization and dual variables that control the layer-wise pruning ratio. Although these methods incorporate resource consumption as a constraint in the training process, the range of attainable budgets is limited by the depth of the model. Besides, generating data measurements to model resource consumption per hardware and architecture can be expensive especially on low-end hardware platforms.

Layer pruning. Unlike filter pruning, little attention is paid to shallow CNNs in the pruning literature. In SSS [37], the authors propose to train a scaling factor for structure selection such as neurons, blocks, and groups. However, shallower models are only possible with architectures with residual connections to allow data flow in the optimization process. Closest to our work for a general (unconstrained by architecture type) layer pruning approach is the work

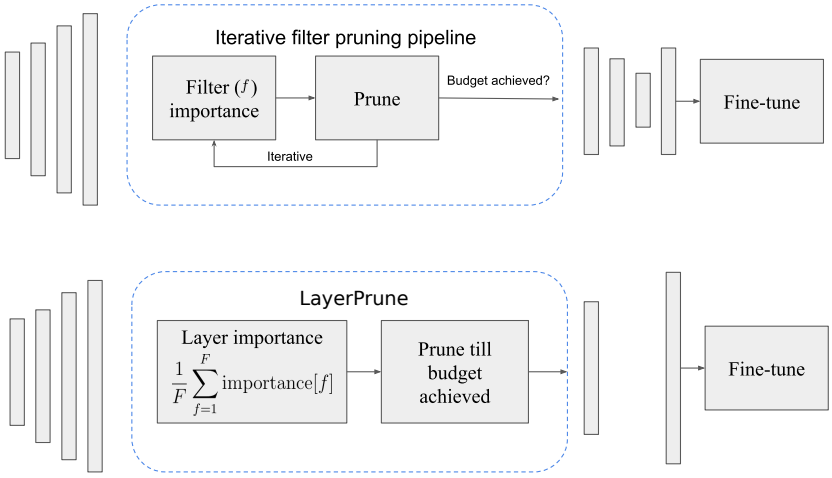

Fig. 3: Main pipeline illustrates the difference between typical iterative filter pruning and proposed LayerPrune framework. Filter pruning (top) produces thinner architecture in an iterative process while LayerPrune (bottom) prunes whole layers in one-shot. In LayerPrune, layer's importance is calculated as the average importance of each filter f in all filters F at that layer.

done by Chen et al. [38]. In their method, linear classifiers probes are utilized and trained independently per layer for layer-ranking. After the layer-ranking learning stage, they prune the least important layers and fine-tune the shallower model. Although [38] requires rank training, it is without any gain in classification accuracy compared to our one-shot LayerPrune layer ranking as will be shown in the experiments section.

3 Methodology

In this section, we describe in detail LayerPrune for layer pruning using existing filter criteria along with a novel layer-wise accuracy approximation. A typical filter pruning method follows a three-stage pipeline as illustrated in Figure 3. Filter importance is iteratively re-evaluated after each pruning step based on a pruning meta-parameter such as pruning N filters or pruning those $\leq$ threshold. In LayerPrune, we remove the need for the iterative pruning step and show that using the same filter criterion, we can remove layers in a one-shot to respect a budget. This simplifies the pruning step to a hyper-parameter free process and is computationally efficient. Layer importance is calculated as the average of filter importance in this layer.

3.1 Statistics-based criterion

Although existing filter pruning methods are different in algorithms and optimization used, they focus more on finding the optimal per-layer number of filters and share common filter criteria. We divide the methods based on the filter criterion used and propose their layer importance counterpart used in LayerPrune.

Preliminary notion. Consider a network with L layers, each layer l has weight matrix $W^{(l)} \in \mathbb{R}^{N_l \times F_l \times K_l \times K_l}$ with N_l input channels, F_l number of filters and K_l is the size of the filters at this channel. Evaluated criteria and methods are:

Weight statistics. [25,33,21] differ in the optimization algorithm but share weight statistics as a filter ranking. Layer pruning for this criteria is calculated as:

$$\text{weights-layer-importance}[l] = \frac{1}{F_l} \sum_{i=1}^{F_l} \left\| W^{(l)}[:, i, :, :] \right\|_2 \tag{1}$$

Taylor weights. Taylor method [24] is slightly different from previous criterion in that the gradients are included in the ranking as well. Filter f ranking is based on $\sum_s (g_s w_s)^2$ where s iterates over all individual weights in f, g is the gradient, w is the weight value. Similarly, layer ranking can be expressed as:

$$\text{taylor-layer-importance}[l] = \frac{1}{F_l} \sum_{i=1}^{F_l} \left\| G^{(l)}[:, i, :, :] \odot W^{(l)}[:, i, :, :] \right\|_2 \tag{2}$$

where $\odot$ is element-wise product and $G^{(l)} \in \mathbb{R}^{N_l \times F_l \times K_l \times K_l}$ is the gradient of loss with respect to weights $W^{(l)}$.

Feature map based heuristics. [23,39,40] rank filters based on statistics from output of layer. In [23], ranking is based on the effect on the next layer while [39], similar to Taylor weights, utilizes gradients and norm but on feature maps.

Channel saliency. In this criterion, a scalar is multiplied by the feature maps and optimized within a typical training cycle with task loss and sparsity regularization loss to encourage sparsity. Slimming [31] utilizes Batch Normalization scale γ as the channel saliency. Similarly, we use Batch Normalization scale parameter to calculate layer importance for this criteria, specifically:

$$\text{BN-layer-importance}[l] = \frac{1}{F_l} \sum_{i=1}^{F_l} (\gamma_i^{(l)})^2 \tag{3}$$

Ensemble. We also consider diverse ensemble of layer ranks where the ensemble rank of each layer is the sum of its rank per method, more specifically:

$$\text{ensemble-rank}[l] = \sum_{m \in \{1 \ldots M\}} (\text{LayerRank}(m, l)) \tag{4}$$

where l is the layer's index, M is the number of all criteria and LayerRank indicates the order of layer l in the sorted list for criterion m.

3.2 Efficiency-based criterion

In addition to existing filter criteria, we present a novel layer importance by layer-wise accuracy approximation. Motivated by the few-shot learning literature [41,42], we use imprinting to approximate the classification accuracy up to each layer. Imprinting is used to approximate a classifier's weight matrix when only a few training samples are available. Although we have adequate training samples, we are inspired by the efficiency of imprinting to approximate the accuracy in one pass without the need for training. We create a classifier proxy for each prunable candidate (e.g convolution layer or residual blocks), and then the training data is used to imprint the classifier weight matrix for each proxy. Since each layer has a different output feature shape, we apply adaptive average pooling to simplify our method and unify the embedding length so that each layer produces roughly an output of the same size. Specifically, the pooling is done as follows:

$$\begin{aligned} d_i &= \text{round}(\sqrt{\frac{N}{n_i}}) \\ E_i &= \text{AdaptiveAvgPool}(O_i, d_i), \end{aligned} \tag{5}$$

where N is the embedding length, n_i is layer i's number of filters, O_i is layer i's output feature map, and AdaptiveAvgPool [43] reduces O_i to embedding $E_i \in \mathbb{R}^{d_i \times d_i \times n_i}$. Finally, embeddings per layer are flattened to be used in imprinting. Imprinting calculates the proxy classifier's weights matrix P_i as follows:

$$P_i[:, c] = \frac{1}{N_c} \sum_{j=1}^{D} \mathbb{I}_{[c_j == c]} E_j \tag{6}$$

where c is the class id, c_j is sample's j class id, N_c is the number of samples in class c, D is the total number of samples, and $\mathbb{I}_{[.]}$ denotes the indicator function.

The accuracy at each proxy is then calculated using the imprinted weight matrices. The prediction for each sample j is calculated for each layer i as:

$$\hat{y}_j = \underset{c \in \{1,\ldots,C\}}{\text{argmax}} \; P_i[:, c]^T E_j, \tag{7}$$

where E_j is calculated as shown in Eq.(5). This is equivalent to finding the nearest class from the imprinted weights in the embedding space. Ranking of each layer is then calculated as the gain in accuracy from previous pruning candidate.

4 Evaluation Results

In this section we present our experimental results comparing state-of-the-art pruning methods and LayerPrune in terms of accuracy and latency reduction

on two different hardware platforms. We show latency on high-end GPU 1080Ti and on NVIDIA Jetson Xavier embedded device, which is used in mobile vision systems and contains 512-core Volta GPU. We evaluate the methods on CIFAR10/100 [44] and ImageNet [1] datasets.

4.1 Implementation details

Latency calculation. Latency model is averaged over 1000 forward pass after 10 warm up forward passes for lazy GPU initialization. Latency is calculated using batch size 1, unless otherwise stated, due to its practical importance in real-time application as in robotics where we process an online stream of frames. All pruned architectures are implemented and measured using PyTorch [45]. For a fair comparison, we compare latency reduction on similar accuracy retention from baseline and reported by original papers or compare accuracy on similar latency reduction with methods supporting layer or block pruning.
Handling filter shapes after layer removal. If the pruned layer l with weight $W^{(l)} \in \mathbb{R}^{N_l \times F_l \times K_l \times K_l}$ has $N_l \neq F_l$, we replace layer $(l+1)$'s weight matrix from $W^{(l+1)} \in \mathbb{R}^{F_l \times F_{l+1} \times K_{l+1} \times K_{l+1}}$ to $W^{(l+1)} \in \mathbb{R}^{N_l \times F_{l+1} \times K_{l+1} \times K_{l+1}}$ with random initialization. All other layers are initialized from the pre-trained dense model.

4.2 Results on CIFAR

We evaluate CIFAR-10 and CIFAR-100 on ResNet56 [4] and VGG19-BN [46].

Random filters vs. Random layers Initial hypothesis verification is to generate random filter and layer pruned models, then train them to compare their accuracy and latency reduction. Random models generation follows the same setup as explained in Section (1). Each model is trained with SGD optimization for 164 epochs with learning rate 0.1 that decays by 0.1 at epochs 81, 121, and 151. Figure 4 shows the latency-accuracy plot for both random pruning methods. Layer pruned models outperform filter pruned ones in accuracy by 7.09% on average and can achieve up to 60% latency reduction. Also, within the same latency budget, filter pruning shows higher variance in accuracy than layer pruning. This suggests that latency constrained optimization with filter pruning is complex and requires careful per layer pruning ratio selection. On the other hand, layer pruning has small accuracy variation, in general within a budget.

VGG19-BN Results on CIFAR-100 are presented in Table 1. The table is divided based on the previously mentioned filter criterion categorization in Section 3.1. First, we compare with Chen et el. [38] on a similar latency reduction as both [38] and LayerPrune perform layer pruning. Although [38] requires training for layer ranking, LayerPrune outperforms it by 1.11%. We achieve up to 56% latency reduction with 1.52% accuracy increase from baseline. As VGG19-BN is over-parametrized for CIFAR-100, removing layers act as a regularization and can find models with better accuracy than the baseline. Unlike with filter pruning

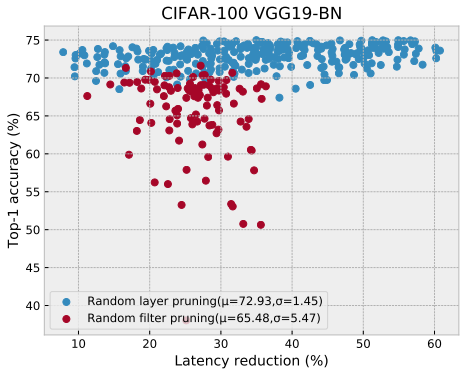

Fig. 4: Random filter pruned and layer pruned models generated from VGG19-BN (Top-1=73.11%). Accuracy mean and standard deviation is shown in parentheses.

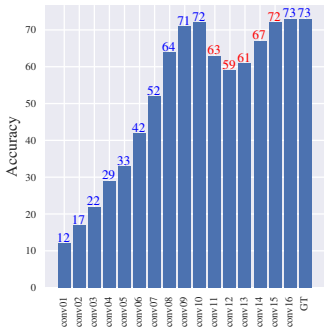

Fig. 5: Layer-wise accuracy using imprinting on CIFAR-100. Red indicates drop in accuracy.

methods, they are bounded by small accuracy variations around the baseline. It is worth mentioning that latency reduction of removing the same number of filters using different filter criteria varies from -0.06% to 40.0%. While layer pruned models, with the same number of pruned layers, regardless of the criterion range from 34.3% to 41%. That suggests that latency reduction using filter pruning is sensitive to environment setup and requires complex optimization to respect a latency budget.

To further explain the accuracy increase by LayerPrune, Fig. 5 shows layer-wise accuracy approximation on baseline VGG19-BN using the imprinting method explained in Section (3.2). Each bar represents the approximated classification accuracy up to this layer (rounded for visualization). We see a drop in accuracy followed by an increasing trend from conv10 to conv15. This is likely because the number of features is the same from conv10 to conv12. We start to observe an accuracy increase only at conv13 that follows a max-pooling layer and has twice as many features. That highlights the importance of downsampling and doubling the number of features at this point in the model. So layer pruning does not only improve inference speed but can also discover a better regularized shallow model especially on a small dataset. It is also worth mentioning that both the proxy classifier from the last layer, conv16, and the actual model classifier, GT, have the same accuracy, showing how the proxy classifier is a plausible approximation of the converged classifier.

ResNet56 We also compare on the more complex architecture ResNet56 on CIFAR-10 and CIFAR-100 in Table 2. On a similar latency reduction, LayerPrune outperforms [38] by 0.54% and 1.23% on CIFAR-10 and CIFAR-100

VGG19 (73.11%)						
Method	Shallower?	Top1 Acc. (%)	1080Ti LR (%)		Xavier LR (%)	
			bs=8	bs=64	bs=8	bs=64
Chen et al. [38]	✓	73.25	56.01	52.86	58.06	49.86
LayerPrune_8-Imprint	✓	**74.36**	56.10	53.67	57.79	49.10
Weight norm [25]	✗	73.01	-2.044	-0.873	-4.256	-0.06
ECC [21]	✗	72.71	16.37	36.70	29.17	**36.69**
LayerPrune_2	✓	73.60	17.32	14.57	19.512	10.97
LayerPrune_5	✓	**74.80**	**39.84**	**37.85**	**41.86**	34.38
Slimming [31]	✗	72.32	16.84	**40.08**	40.55	**39.53**
LayerPrune_2	✓	73.60	17.34	13.86	18.85	10.90
LayerPrune_5	✓	**74.80**	**39.56**	37.30	**41.40**	34.35
Taylor [24]	✗	72.61	15.87	19.77	-4.89	17.45
LayerPrune_2	✓	73.60	17.12	13.54	18.81	10.89
LayerPrune_5	✓	**74.80**	**39.36**	**37.12**	**41.34**	**34.44**

Table 1: **Comparison of different pruning methods on VGG19-BN CIFAR-100.** The accuracy for baseline model is shown in parentheses. LR, bs stands for latency reduction and batch size respectively. x in LayerPrune_x indicates number of layers removed. -ve LR indicates increase in latency. Shallower indicates whether a method prunes layers. Best is shown in **bold**.

respectively. On the other hand, within each filter criterion, LayerPrune outperforms filter pruning and is on par with the baseline in accuracy. In addition, filter pruning can result in latency increase (i.e negative LR) with specific hardware targets and batch sizes [47] as shown with batch size 8. However, LayerPrune consistently shows latency reduction under different environmental setups. We also compare with larger batch size to further encourage filter pruned models to better utilize the resources. Still, we found LayerPrune achieves overall better latency reduction with a large batch size. Latency reduction variance, LR var, between different batch sizes within the same hardware platform is shown as well. Consistent with previous results on VGG, LayerPrune is less sensitive to changes in criterion, batch size, and hardware than filter pruning. We also show results up to 2.5x latency reduction with less than 2% accuracy drop.

4.3 Results on ImageNet

We evaluate the methods on ImageNet dataset for classification. For all experiments in this section, PyTorch pre-trained models are used as a starting point for network pruning. We follow the same setup as in [24] where we prune 100 filters each 30 mini-batches for 10 pruning iterations. The pruned model is then fine-tuned with learning rate $1e^{-3}$ using SGD optimizer and 256 batch size. Results on ResNet50 are presented in Table 3. In general, LayerPrune models improve accuracy over the baseline and their counterpart filter pruning methods. Although feature maps criterion [39] achieves better accuracy by 0.92% over LayerPrune_1, LayerPrune has higher latency reduction that exceeds by 5.7%.

Method	Shallower?	Top1 Acc. (%)	1080Ti LR (%)		Xavier LR (%)	
			bs=8	bs=64	bs=8	bs=64
CIFAR-10 ResNet56 baseline (93.55%)						
Chen et al. [38]	✓	93.09	**26.60**	**26.31**	26.96	25.66
LayerPrune$_8$-Imprint	✓	**93.63**	**26.41**	**26.32**	**27.30**	**29.11**
Taylor weight [24]	✗	93.15	0.31	5.28	-0.11	2.67
LayerPrune$_1$	✓	**93.49**	2.864	**3.80**	5.97	5.82
LayerPrune$_2$	✓	93.35	**6.46**	**8.12**	**9.33**	**11.38**
Weight norm [25]	✗	92.95	-0.90	5.22	1.49	3.87
L1 norm [33]	✗	93.30	-1.09	-0.48	2.31	1.64
LayerPrune$_1$	✓	**93.50**	2.72	3.88	7.08	5.67
LayerPrune$_2$	✓	93.39	**5.84**	**7.94**	**10.63**	**11.45**
Feature maps [39]	✗	92.7	-0.79	6.17	1.09	**8.38**
LayerPrune$_1$	✓	**92.61**	3.29	2.40	7.77	2.76
LayerPrune$_2$	✓	92.28	**6.68**	**5.63**	**11.11**	5.05
Batch Normalization [31]	✗	93.00	0.6	3.85	2.26	1.42
LayerPrune$_1$	✓	**93.49**	2.86	3.88	7.08	5.67
LayerPrune$_2$	✓	93.35	**6.46**	**7.94**	**10.63**	**11.31**
LayerPrune$_{18}$-Imprint	✓	92.49	57.31	55.14	57.57	63.27
CIFAR-100 ResNet56 baseline (71.2%)						
Chen et al. [38]	✓	69.77	**38.30**	34.31	38.53	39.38
LayerPrune$_{11}$-Imprint	✓	**71.00**	**38.68**	**35.83**	**39.52**	**54.29**
Taylor weight [24]	✗	71.03	2.13	5.23	-1.1	3.75
LayerPrune$_1$	✓	**71.15**	3.07	3.74	3.66	5.50
LayerPrune$_2$	✓	70.82	**6.44**	**7.18**	**7.30**	**11.00**
Weight norm [25]	✗	71.00	2.52	6.46	-0.3	3.86
L1 norm [33]	✗	70.65	-1.04	4.06	0.58	1.34
LayerPrune$_1$	✓	**71.26**	3.10	3.68	4.22	5.47
LayerPrune$_2$	✓	71.01	**6.59**	**7.03**	**8.00**	**10.94**
Feature maps [39]	✗	70.00	1.22	9.49	-1.27	**7.94**
LayerPrune$_1$	✓	**71.10**	2.81	3.24	4.46	5.56
LayerPrune$_2$	✓	70.36	**6.06**	**6.70**	**7.72**	**7.85**
Batch Normalization [31]	✗	70.71	0.37	2.26	-1.02	2.89
LayerPrune$_1$	✓	**71.26**	3.10	3.68	4.22	5.47
LayerPrune$_2$	✓	70.97	**6.36**	**6.78**	**7.59**	**10.94**
LayerPrune$_{18}$-Imprint	✓	68.45	60.69	57.15	61.32	71.65

Table 2: **Comparison of different pruning methods on ResNet56 CIFAR-10/100.** The accuracy for baseline model is shown in parentheses. LR and bs stands for latency reduction and batch size respectively. subscript x in LayerPrune$_x$ indicates number of blocks removed.

It is worth mentioning that the latency aware optimization ECC has an upper bound latency reduction of 11.56%, on 1080Ti, with accuracy 16.3%. This stems from the fact that iterative filter pruning is bounded by the network's depth and structure dependency within the network, thus not all layers are considered for pruning such as the gates at residual blocks. Besides, ECC builds a layer-wise bilinear model to approximate the latency of a model given the number of input channels and output filters per layer. This simplifies the non-linear

relationship between the number of filters per layer and latency. We show the latency reduction on Xavier for an ECC pruned model optimized for 1080Ti, and this pruned model results in a latency increase on batch size 1 and the lowest latency reduction on batch size 64. This suggests that a hardware-aware filter pruned model for one hardware architecture might perform worse on another hardware than even a hardware-agnostic filter pruning method. It is worth noting that the filter pruning HRank [40] with 2.6x FLOPs reduction shows large accuracy degradation compared to LayerPrune (71.98 vs 74.31). Even with aggressive filter pruning, speed up is noticeable with large batch size but shows small speed gain with small batch size. Within shallower models, LayerPrune outperforms SSS on the same latency budget even when SSS supports block pruning for ResNet50, which shows the effectiveness of accuracy approximation as layer importance.

ResNet50 baseline (76.14)						
Method	Shallower?	Top1 Acc. (%)	1080Ti LR (%)		Xavier LR (%)	
			bs=1	bs=64	bs=1	bs=64
Batch Normalization	✗	75.23	2.49	1.61	-2.79	4.13
LayerPrune$_1$	✓	**76.70**	15.95	4.81	21.38	6.01
LayerPrune$_2$	✓	76.52	**20.41**	**8.36**	**25.11**	**9.96**
Taylor [24]	✗	76.4	2.73	3.6	-1.97	6.60
LayerPrune$_1$	✓	**76.48**	15.79	3.01	21.52	4.85
LayerPrune$_2$	✓	75.61	**21.35**	**6.18**	**27.33**	**8.42**
Feature maps [39]	✗	**75.92**	10.86	3.86	20.25	8.74
Channel pruning* [48]	✗	72.26	3.54	6.13	2.70	7.42
ThiNet* [23]	✗	72.05	10.76	**10.96**	15.52	**17.06**
LayerPrune$_1$	✓	75.00	16.56	2.54	23.82	4.49
LayerPrune$_2$	✓	71.90	**22.15**	5.73	**29.66**	8.03
SSS-ResNet41 [37]	✓	75.50	25.58	24.17	31.39	21.76
LayerPrune$_3$-Imprint	✓	**76.40**	22.63	25.73	30.44	20.38
LayerPrune$_4$-Imprint	✓	75.82	**30.75**	**27.64**	**33.93**	**25.43**
SSS-ResNet32 [37]	✓	74.20	**41.16**	29.69	**42.05**	29.59
LayerPrune$_6$-Imprint	✓	**74.74**	40.02	**36.59**	41.22	34.50
HRank-2.6x-FLOPs* [40]	✗	71.98	11.89	36.09	20.63	**40.09**
LayerPrune$_7$-Imprint	✓	**74.31**	**44.26**	**41.01**	**41.01**	38.39

Table 3: **Comparison of different pruning methods on ResNet50 ImageNet.** * manual pre-defined signatures. ** same pruned model optimized for 1080Ti latency consumption model in ECC optimization

4.4 Layer pruning comparison

In this section, we analyze different criteria for layer pruning under the same latency budget as presented in Table 4. Our imprinting method consistently outperforms other methods, especially on higher latency reduction rates. Imprinting is able to get 30% latency reduction with only 0.36% accuracy loss from

ResNet50 (76.14)				
	1 block (LR≈ 15%)	2 blocks (LR≈ 20%)	3 blocks (LR≈ 25%)	4 blocks (LR≈ 30%)
LayerPrune-Imprint	**76.72**	**76.53**	**76.40**	**75.82**
LayerPrune-Taylor	76.48	75.61	75.34	75.28
LayerPrune-Feature map	75.00	71.9	70.84	69.05
LayerPrune-Weight magnitude	**76.70**	**76.52**	76.12	74.33
LayerPrune-Batch Normalization	**76.70**	76.22	75.84	75.03
LayerPrune-Ensemble	**76.70**	76.11	75.76	75.01

Table 4: **Comparison of different layer pruning methods supported by LayerPrune on ResNet50 ImageNet.** Latency reduction is calculated on 1080Ti with batch size 1.

baseline. The ensemble method, although has better accuracy than the average accuracy, is still sensitive to individual errors. We further compare layer pruning by imprinting on a similar latency budget with smaller ResNet variants. We outperform ResNet34 by 1.44% (LR=39%) and ResNet18 by 0.56% (LR=65%) in accuracy showing the effectiveness of incorporating accuracy in block importance. Detailed numerical evaluation can be found in supplementary.

5 Conclusion

We presented LayerPrune framework which includes a set of layer pruning methods. We show the benefits of LayerPrune on latency reduction compared to filter pruning. The key findings of this paper are the following:

- For a filter criterion, training a LayerPrune model based on this criterion achieves the same, if not better, accuracy as the filter pruned model obtained by using the same criterion.
- Filter pruning compresses the number of convolution operations per layer and thus latency reduction depends on hardware architecture, while LayerPrune removes the whole layer. As result, filter pruned models might produce non-optimal matrix shapes for the compute kernels that can lead even to latency increase on some hardware targets and batch sizes.
- Filter pruned models within a latency budget have a larger variance in accuracy than LayerPrune. This stems from the fact that the relation between latency and number of filters is non-linear and optimization constrained by a resource budget requires complex per-layer pruning ratios selection.
- We also showed the importance of incorporating accuracy approximation in layer ranking by imprinting.

Acknowledgment

We thank Compute Canada and WestGrid for their supercomputers to conduct our experiments.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105
2. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
3. Elkerdawy, S., Zhang, H., Ray, N.: Lightweight monocular depth estimation model by joint end-to-end filter pruning. In: 2019 IEEE International Conference on Image Processing (ICIP), IEEE (2019) 4290–4294
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE CVPR. (2016) 770–778
5. Wu, Z., Shen, C., Van Den Hengel, A.: Wider or deeper: Revisiting the resnet model for visual recognition. Pattern Recognition (2019) 119–133
6. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European conference on computer vision (ECCV). (2018) 116–131
7. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
8. Wang, R.J., Li, X., Ling, C.X.: Pelee: A real-time object detection system on mobile devices. In: Advances in Neural Information Processing Systems. (2018) 1963–1972
9. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2017) 4700–4708
10. Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., Weinberger, K.Q.: Multi-scale dense networks for resource efficient image classification. arXiv preprint arXiv:1703.09844 (2017)
11. Kurt Keutzer, e.a.: Abandoning the dark arts: Scientific approaches to efficient deep learning. The 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing , Conference on Neural Information Processing Systems (2019)
12. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: Haq: Hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE CVPR. (2019) 8612–8620
13. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research **18** (2017) 6869–6898
14. Yang, C., Xie, L., Su, C., Yuille, A.L.: Snapshot distillation: Teacher-student optimization in one generation. In: Proceedings of the IEEE CVPR. (2019) 2859–2868
15. Jin, X., Peng, B., Wu, Y., Liu, Y., Liu, J., Liang, D., Yan, J., Hu, X.: Knowledge distillation via route constrained optimization. In: Proceedings of the IEEE ICCV. (2019) 1345–1354
16. Yang, T.J., Chen, Y.H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2017) 5687–5695
17. Yang, T.J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: Netadapt: Platform-aware neural network adaptation for mobile applications. In: Proceedings of the ECCV. (2018) 285–300

18. Bianco, S., Cadene, R., Celona, L., Napoletano, P.: Benchmark analysis of representative deep neural network architectures. IEEE Access (2018) 64270–64277
19. van Werkhoven, B.: Kernel tuner: A search-optimizing gpu code auto-tuner. Future Generation Computer Systems (2019) 347–358
20. Nugteren, C., Codreanu, V.: Cltune: A generic auto-tuner for opencl kernels. In: 2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip, IEEE (2015) 195–202
21. Yang, H., Zhu, Y., Liu, J.: Ecc: Platform-independent energy-constrained deep neural network compression via a bilinear regression model. In: Proceedings of the IEEE CVPR. (2019) 11206–11215
22. Lemaire, C., Achkar, A., Jodoin, P.M.: Structured pruning of neural networks with budget-aware regularization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2019) 9108–9116
23. Luo, J.H., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. In: Proceedings of the IEEE ICCV. (2017) 5058–5066
24. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE CVPR. (2019) 11264–11272
25. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. (2015) 1135–1143
26. Han, S., Mao, H., Dally, W.: Compressing deep neural networks with pruning, trained quantization and huffman coding. ICLR 2017 (2015)
27. Srinivas, S., Babu, R.V.: Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149 (2015)
28. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. arXiv preprint arXiv:1712.01312 (2017)
29. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018)
30. Sharify, S., Lascorz, A.D., Mahmoud, M., Nikolic, M., Siu, K., Stuart, D.M., Poulos, Z., Moshovos, A.: Laconic deep learning inference acceleration. In: Proceedings of the 46th International Symposium on Computer Architecture. (2019) 304–317
31. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE ICCV. (2017) 2736–2744
32. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems. (2016) 2074–2082
33. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. ICLR (2017)
34. Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W., Tian, Q.: Variational convolutional neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2019) 2780–2789
35. Chin, T.W., Zhang, C., Marculescu, D.: Layer-compensated pruning for resource-constrained convolutional neural networks. NeurIPS (2018)
36. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: Proceedings of the ECCV). (2018) 784–800
37. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. In: Proceedings of the European conference on computer vision (ECCV). (2018) 304–320

38. Chen, S., Zhao, Q.: Shallowing deep networks: Layer-wise pruning based on feature representations. IEEE transactions on pattern analysis and machine intelligence (2018) 3048–3056
39. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. arXiv preprint arXiv:1611.06440 **3** (2016)
40. Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L.: Hrank: Filter pruning using high-rank feature map. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. (2020) 1529–1538
41. Qi, H., Brown, M., Lowe, D.G.: Low-shot learning with imprinted weights. In: Proceedings of the IEEE CVPR. (2018) 5822–5830
42. M. Siam, B.O., Jagersand, M.: Amp: Adaptive masked proxies for few-shot segmentation. In: Proceedings of the IEEE ICCV. (2019)
43. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: ECCV, Springer International Publishing (2014) 346–361
44. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. (2009)
45. Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. The Journal of Machine Learning Research (2017) 5595–5637
46. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. ICLR (2015)
47. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE **105** (2017) 2295–2329
48. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE ICCV. (2017) 1389–1397