ADAPTIVE ACTIVATION-BASED STRUCTURED PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Pruning is a promising approach to compress complex deep learning models in order to deploy them on resource-constrained edge devices. However, many existing pruning solutions are based on unstructured pruning, which yield models that cannot efficiently run on commodity hardware, and require users to manually explore and tune the pruning process, which is time consuming and often leads to sub-optimal results. To address these limitations, this paper presents an adaptive, activation-based, structured pruning approach to automatically and efficiently generate small, accurate, and hardware-efficient models that meet user requirements. First, it proposes iterative structured pruning using activation-based attention feature maps to effectively identify and prune unimportant filters. Then, it proposes adaptive pruning policies for automatically meeting the pruning objectives of accuracy-critical, memory-constrained, and latency-sensitive tasks. A comprehensive evaluation shows that the proposed method can substantially outperform the state-of-the-art structured pruning works on CIFAR-10 and ImageNet datasets. For example, on ResNet-56 with CIFAR-10, without any accuracy drop, our method achieves the largest parameter reduction (79.11%), outperforming the related works by 22.81% to 66.07%, and the largest FLOPs reduction (70.13%), outperforming the related works by 14.13% to 26.53%.

1 INTRODUCTION

Deep neural networks (DNNs) have substantial compute and memory requirements. As deep learning becomes pervasive and moves towards edge devices, DNN deployment becomes harder because of the mistmatch between resource-hungry DNNs and resource-constrained edge devices. Many techniques have been proposed to alleviate this resource pressure (Chu et al. (2003); Han et al. (2015); Polino et al. (2018); Yim et al. (2017); Pham et al. (2018)). Among these efforts, DNN pruning is a promising approach (Li et al. (2016); Han et al. (2015); Molchanov et al. (2016); Theis et al. (2018); Renda et al. (2020)), which identifies the parameters (or weight elements) that do not contribute significantly to the accuracy and prunes them from the network.

Unstructured vs. Structured Pruning. Pruning techniques can be broadly categorized into two classes, namely, unstructured pruning and structured pruning, which take different approaches to prune a model and have different impact to the performance of the pruned model.

Unstructured pruning (LeCun et al. (1990); Han et al. (2015); Molchanov et al. (2017)) is a fine-grained approach that prunes individual unimportant elements in weight tensors. It has less impact to model accuracy, compared to structured pruning, as it selectively prunes unimportant elements from a weight filter while retaining the important elements. However, research has shown that traditional sparse computation libraries, like cuSPARSE or MKLDNN, observe substantial slowdown on unstructured pruned models because the libraries are designed for extremely high sparsity (>99%), whereas unstructured sparsity leads to much lower (75%-90%) sparsity (Hill et al. (2017)). To solve this, there are initial efforts to manually write kernels suitable for this sparsity (Elsen et al. (2019)). But overall, unstructured pruned models are still hard to accelerate on commodity hardware.

Structured pruning is a coarse-grained approach that prunes entire regular regions of weight tensors according to some rule-based heuristics, such as L1-norm (Li et al. (2016)), average percentage of zero (Molchanov et al. (2016)), and other information considering the relationship between neighboring layers (Theis et al. (2018); Lee et al. (2018)). Compared to unstructured pruning, it is more

difficult to prune a model without causing accuracy loss using structured pruning, because by removing entire regions, it might remove weight elements that are important to the final accuracy (Li et al. (2016)). However, structured pruned models can be mapped easily to general-purpose hardware and accelerated directly with off-the-shelf hardware and libraries (He et al. (2018b)).

One Shot vs. Iterative Pruning. Another choice in DNN pruning is between one-shot pruning and iterative pruning. One-shot pruning prunes a pre-trained model and then retrains it once to make up for the accuracy loss. Iterative pruning prunes and retrains the model in multiple rounds. Both techniques can choose either structured or unstructured pruning techniques. Recently, works based on the Lottery Ticket Hypothesis (LTH) have achieved great successes in creating smaller and more accurate models through iterative pruning with rewinding (Frankle & Carbin (2018)). LTH posits that a dense randomly initialized network has a sub-network, termed as a *winning ticket*, which can achieve an accuracy comparable to the original network. At the beginning of each pruning round, it rewinds the weights and/or learning rate of the sub-network to some early epoch of the training phase of the original model to reduce the distance between the sub-network and original model and increase the change of finding the winning ticket. However, most of LTH-based works considered only unstructured pruning, e.g., Iterative Magnitude Pruning (IMP) (Frankle & Carbin (2018); Frankle et al. (2019)), which, as discussed above, is hardware-inefficient.

It is non-trivial to design an iterative pruning method with structured pruning. To understand the state of iterative structured pruning, we experimented with IMP's structured pruning counterpart—L1-norm structured pruning (ILP) (Li et al. (2016)) which removes entire filters depending on their L1-norm value. We observed that ILP cannot effectively prune a model while maintaining its accuracy, e.g., ILP can prune ResNet-50 by at most 11.5% of parameters when the maximum accuracy loss is limited to 1% on ImageNet. Therefore, directly applying iterative pruning with existing weight-magnitude based structured pruning methods does not produce accurate pruned models. In this paper, we study how to produce *small*, *accurate*, and *hardware-efficient* models based on *iterative structured pruning with rewinding*.

Automatic Pruning. For pruning to be useful in practice, it is important to automatically meet the pruning objectives for diverse ML applications and devices. Most pruning methods require users to explore and adjust multiple hyper-parameters, e.g., with LTH-based iterative pruning, users need to determine how many parameters to prune in each round. Tuning the pruning process is time consuming and often leads to sub-optimal results. Ultimately, users only care about model accuracy, size, and inference speed. For example, user may want to minimize the model size while keeping the accuracy loss to below an acceptable threshold, or to minimize the accuracy loss while ensuring the model size fits the device memory or the inference latency meets the application requirement. Therefore, we study how to *automatically* adapt the pruning process in order to *meet the pruning objectives without user intervention*.

Some works (Zoph et al. (2018); Cai et al.; Ashok et al. (2017)) use reinforcement-learning algorithms to find computationally efficient architectures. These works are in the area of Neural Architecture Search (NAS), among which the most recent is AutoML for Model Compression (AMC) (He et al. (2018b)). AMC enables the model to arrive at the target speedup by limiting the action space (the sparsity ratio for each layer), and it finds out the limit of compression that offers no loss of accuracy by tweaking the reward function. But this method has to explore over a large search space of all available layer-wise sparsity, which is time consuming when neural networks are large and datasets are complicated. More recently, Zhuang et. al. proposed Polarization-based structured pruning (Zhuang et al. (2020)) to prune a model automatically given a target reduced percentage of floating point operations (FLOPs). This work has several limitations: 1) it does not consider other important pruning objectives such as model accuracy and size; 2) it prunes filters based on their weight values, which as explained above cannot effectively capture the filters' importance; and 3) its search algorithm is empirically tuned and cannot guarantee the optimal outcome.

Our Solution. We propose *activation-based*, *adaptive*, *iterative structured pruning* to find the "winning ticket" models that are at the same time hardware efficient and to automatically meet the users' model accuracy, size, and speed requirements.

First, we propose an activation-based structured pruning method to identify and remove unimportant filters in an LTH-based iterative pruning (with rewinding) process. Specifically, we properly define an attention mapping function that takes a 2D activation feature maps of a filter as input, and outputs

Algorithm 1 Adaptive Iterative Structured Pruning Algorithm

1: [Initialize] Initialize a network $f(x; M^0 \odot W_0^0)$ with initial mask $M^0 = \{0,1\}^{|W_0^0|}$ and Threshold $T[0]$
2: [Save weights] Train the network for k epochs, yielding network $f(x; M^0 \odot W_k^0)$, and save weights W_k^0
3: [Train to converge] Train the network for $T-k$ epochs to converge, producing network $f(x; M^0 \odot W_T^0)$
4: **for** pruning round r ($r \geqslant 1$) **do**
5: [Prune] Prune filters from W_T^{r-1} using $T[r]$, producing a mask M^r, and a network $f(x; M^r \odot W_T^{r-1})$
6: [Rewind Weights] Reset the remaining filters to W_k^0 at epoch k, producing network $f(x; M^r \odot W_k^{r-1})$
7: [Rewind Learning Rate] Reset the learning rate schedule to its state from epoch k
8: [Retrain] Retrain the unpruned filters for $T-k$ epoch to converge, yielding network $f(x; M^r \odot W_T^r)$
9: [Evaluate] Evaluate the retrained network $f(x; M^r \odot W_T^r)$ according to the target.
10: [Reset Weights] If the target is not met, reset the weights to an earlier round
11: [Adapt Threshold] Calculate Threshold $T[r+1]$ for the next pruning round
12: **end for**

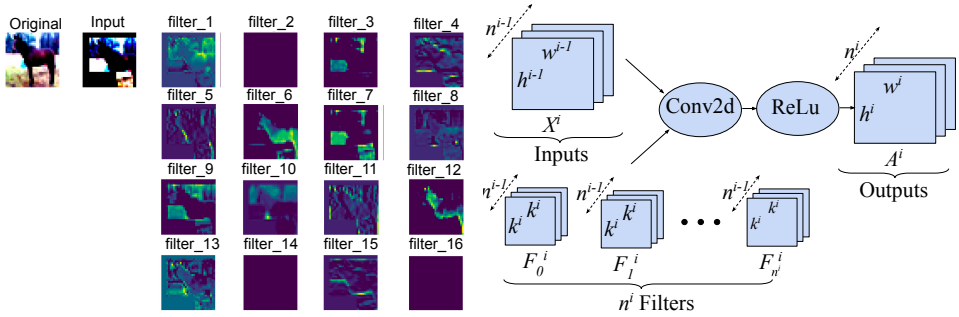
a 1D value used to indicate the importance of the filter. This approach is more effective than weight-value based filter pruning because activation-based attention values not only capture the features of inputs but also contain the information of convolution layers that act as feature detectors for prediction tasks. We then integrate this attention-based method into the LTH-based iterative pruning framework to prune the filters in each round and find the winning ticket that is small, accurate, and hardware-efficient.

Second, we propose adaptive pruning that automatically optimizes the pruning process according to different user objectives. For latency-sensitive scenarios like interactive virtual assistants, we propose FLOPs-guaranteed pruning to achieve the best accuracy given the maximum amount of compute FLOPs; For memory-limited environments like embedded systems, we propose model-size-guaranteed pruning to achieve the best accuracy given the maximum amount of memory footprint; For accuracy-critical applications such as those on self-driving cars, we propose accuracy-guaranteed pruning to create the most resource-efficient model given the acceptable accuracy loss. Aiming for different targets, our method adaptively controls the pruning aggressiveness by adjusting the global threshold used to prune filters. Moreover, it considers the difference in each layer's contributions to the model's size and computational complexity and uses a per-layer threshold, calculated by dividing each layer's remaining parameters or FLOPs by the entire model's remaining parameters or FLOPs, to prune each layer with differentiated level of aggressiveness.

Our results outperform the related works significantly in all cases targeting accuracy loss, parameters reduction, and FLOPs reduction. For example, on ResNet-56 with CIFAR-10 dataset, without accuracy drop, our method achieves the largest parameter reduction (79.11%), outperforming the related works by 22.81% to 66.07%, and the largest FLOPs reduction (70.13%), outperforming the related works by 14.13% to 26.53%. In addition, our method enables a pruned model the reach 0.6% or 1.08% higher accuracy than the original model but with only 30% or 50%of the original's parameters. On ResNet-50 on ImageNet, for the same level of parameters and FLOPs reduction, our method achieves the smallest accuracy loss, lower than the related works by 0.08% to 3.21%; and for the same level of accuracy loss, our method reduces significantly more parameters (6.45% to 29.61% higher than related works) and more FLOPs (0.82% to 17.2% higher than related works).

2 METHODOLOGY

Algorithm 1 illustrates the overall flow of the proposed adaptive activation-based structured pruning. To represent pruning of weights, we use a mask $M^r \epsilon \{0,1\}^d$ for each weight tensor $W_t^r \epsilon R^d$, where r is the pruning round number and t is the training epoch. Therefore, the pruned network at the end of training epoch T is represented by the element-wise product $M^r \odot W_T^r$. The first three steps are to train the original model to completion, while saving the weights at Epoch k. Steps 4–10 represent a pruning round. Step 4 prunes the model (discussed in Section 2.1). Step 5 (optional) and 6 perform rewinding. Step 7 retrains the pruned model for the remaining $T-k$ epochs. Step 8 evaluates the pruned model according to the pruning target. If the target is not met, Step 9 resets the weights to an earlier round. Step 10 calculates the pruning threshold for the next pruning round following the adaptive pruning policy (discussed in Section 2.2).



(a) Filter activation outputs (b) Convolution layer tensors

Figure 1: Illustration of the (a) activation outputs of 16 filters of a conv2d layer, and (b) input and output tensors of a convolution layer. In (a), the left portion shows the original image and the image after data augmentation; and the right portion visualizes the activation outputs of each filer.

2.1 Activation-based Filter Pruning

To realize iterative structured pruning, one can start with the-state-of-the-art structured pruning methods and apply it iteratively. The widely used structured pruning method—L1-norm based structured pruning Renda et al. (2020) removes filters with the lowest L1-norm values, whereas the most recent method—Polarization-based structured pruning Zhuang et al. (2020) improves it by using a regularizer on scaling factors of filters and pruning filters whose scaling factors are below than a threshold. These two methods both assume that the *weight values* of a filter can be used as an indicator about the importance of that filter, much like how LTH uses weight values in unstructured pruning. However, we observe that weight-based structured pruning methods cannot produce accurate pruned models. For example, to prune ResNet-56 on CIFAR-10 with no loss in top-1 accuracy, L1-norm based structured pruning can achieve at most only $1.15\times$ model compression, and Polarization-norm based structured pruning can achieve at most only $1.89\times$ inference speedup. The reason is that, some filters, even though their weight values are small, can still produce useful non-zero activation values that are important for learning features during backpropagation. That is, filters with small values may have large activations.

We propose that the *activation values* of filters are more effective in finding unimportant filters to prune. Activations like ReLu enable non-linear operations, and enable convolutional layers to act as feature detectors. If an activation value is small, then its corresponding feature detector is not important for prediction tasks. So activation values, i.e., the intermediate output tensors after the non-linear activation, not only detect features of training dataset, but also contain the information of convolution layers that act as feature detectors for prediction tasks. We present a visual motivation in Figure 1(a). The figure shows the activation output of 16 filters of a convolution layer on one input image. The first image on the left is the original image, and the second image is the input features after data augmentation. We observe that some filters extract image features with high activation patterns, e.g., the 6th and 12th filters. In comparison, the activation outputs of some filters are close to zero, such as the 2nd, 14th, and 16th. Therefore, from visual inspection, removing filters with weak activation patterns is likely to have low impact on the final accuracy of the pruned model.

There is a natural connection between our activation-based pruning approach and the related attention-based knowledge transfer works (Zagoruyko & Komodakis (2016)). Attention is a mapping function used to calculate the statistics of each element in the activations over the channel dimension. By minimizing the difference of the activation-based attention maps from intermediate layers between a teacher model and a student model, attention transfer enables the student to imitate the behaviour of the teacher. Our proposed pruning method builds upon the same theory that activation-based attention is a good indicator of filters regarding their ability to capture features, and it addresses the new challenges in using activations to guide automatic structured pruning.

In the following, we describe how to prune filters based on its activation feature maps in each round of the iterative structured pruning process. Figure 1(b) shows the inputs and outputs of a 2D convolution layer (referred to as conv2d), followed by an activation layer. For the ith conv2d

layer, let $X^i \epsilon R^{n^{i-1} \times h^{i-1} \times w^{i-1}}$ denote the input features, and $F_j^i \epsilon R^{n^{i-1} \times k^i \times k^i}$ be the jth filter, where h^{i-1} and w^{i-1} are the height and width of the input features, respectively, n^{i-1} is the number of input channels, n^i is the number of output channels, and k^i is the kernel size of the filter. The activation of the jth filter F_j^i after ReLu mapping is therefore denoted by $A_j^i \epsilon R^{h^i \times w^i}$.

The attention mapping function takes a 2D activation $A_j^i \epsilon R^{h^i \times w^i}$ of filter F_j^i as input, and outputs a 1D value which will be used as an indicator of the importance of filters. We consider three forms of activation-based attention maps, where $p \geqslant 1$ and $a_{k,l}^i$ denotes every element of A_j^i: 1) Mean of attention values raised to the power of p: $F_{mean}(A_j^i) = \frac{1}{h^i \times w^i} \sum_{k=1}^{h^i} \sum_{l=1}^{w^i} \left| a_{k,l}^i \right|^p$; 2) Max of attention values raised to the power of p: $F_{max}(A_j^i) = max_{l=1, h^i \times w^i} \left| a_{k,l}^i \right|^p$; and 3) Sum of attention values raised to the power of p: $F_{sum}(A_j^i) = \sum_{k=1}^{h^i} \sum_{l=1}^{w^i} \left| a_{k,l}^i \right|^p$.

Different attention mapping functions defined above can have slightly different properties: 1) Compared to $F_{sum}(A_j^i)$, $F_{mean}(A_j^i)$ eliminates the effect of different heights and widths of output features from different layers. For example, on ResNet-50, the size of each output feature is 112×112 and 56×56 from the first and second convolution layers, respectively, given the input image size is 256×256. Since we aim to prune filters with smaller attention values, $F_{sum}(A_j^i)$ will biasely lead to pruning filters in the last several layers where the size of the filter output feature is smaller. It is not proper that the output size affects the importance of the filter. 2) $F_{max}(A_j^i)$ considers only one of the activation values of each filter, losing information from the other activation elements. Therefore, we choose $F_{mean}(A_j^i)$ as the indicator of the importance of the filter. We have also studied the choice of p (see Appendix A.1), and our results show that, compared to setting p to a higher value, $p = 1$ provides more differentiation between the useful filters and the rest and allows them to stay in the network during the pruning process. Therefore, we choose $F_{mean}(A_j^i)$ with p equals to 1 as the indicator to identify and prune unimportant filters, and our method removes the filters whose attention value is lower than the pruning threshold. The following sections explain how this threshold is automatically adapted in the iterative pruning process and adjusted for each layer based on its contribution to model size and complexity, according to the given pruning objective.

2.2 Adaptive Iterative Pruning

Our approach to pruning is to automatically and efficiently generate a pruned model that meets the users' different objectives. Automatic pruning means that users do not have to figure out how to configure the pruning process. Efficient pruning means that the pruning process should produce the user-desired model as quickly as possible. Users' pruning objectives can vary depending on the usage scenarios: 1) Accuracy-critical tasks, like those used by self-driving cars have stringent accuracy requirement, which is critical for safety, but do not have strict limits on their computing and storage usages; 2) Memory-constrained tasks, like those deployed on microcontrollers have very limited available memory to store the models but do not have strict accuracy requirement; and 3) Latency-sensitive tasks, like those employed by virtual assistants where timely responses are desirable but accuracy is not a hard constraint.

In order to achieve automatic and efficient structured pruning, we propose three adaptive pruning policies to provide 1) Accuracy-guaranteed Adaptive Pruning which produces the most resource-efficient model with the acceptable accuracy loss; 2) Memory-constrained Adaptive Pruning which generates the most accurate model within a given memory footprint; and 3) FLOPs-constrained Adaptive Pruning which creates the most accurate model within a given computational intensity. Specifically, our proposed adaptive pruning method automatically adjusts the global threshold (T), used in our iterative structured pruning algorithm (Algorithm 1) to quickly find the model that meets the pruning objective. Other objectives (e.g., limiting a model's energy consumption) can also be supported by simply plugging in the relevant metrics into our adaptive pruning algorithm. We take Accuracy-guaranteed Adaptive Pruning, described in Algorithm 2, as an example to show the procedure of adaptive pruning. Memory-constrained and FLOPs-constrained Adaptive Pruning algorithms are included in Appendix A.2 and A.3, respectively.

In Algorithm 2, the objective is to guarantee the accuracy loss while minimizing the model size (the algorithm for minimizing the model FLOPs is similar). In the algorithm, T controls the ag-

Algorithm 2 Accuracy-guaranteed Adaptive Pruning

1: **Input:** Target Accuracy Loss $AccLossTarget$
2: **Output:** A small pruned model with an acceptable accuracy
3: Initialize: $T = 0.0, \lambda = 0.005$.
4: **for** pruning round r ($r \geqslant 1$) **do**
5: Prune the model using $T[r]$ (Refer to Algorithm 3)
6: Train the pruned model, evaluate its accuracy $Acc[r]$
7: Calculate the accuracy loss $AccLoss[r]$: $AccLoss[r] = Acc[0] - Acc[r]$
8: **if** $AccLoss[r] < AccLossTarget$ **then**
9: **if** the changes of model size are within 0.1% for several rounds **then**
10: Terminate
11: **else**
12: $\lambda[r+1] = \lambda[r]$
13: $T[r+1] = T[r] + \lambda[r+1]$
14: **end if**
15: **else**
16: Find the last acceptable round k
17: **if** k has been used to roll back for several times **then**
18: Mark k as unacceptable
19: Go to Step 15
20: **else**
21: Roll back model weights to round k
22: $\lambda[r+1] = \lambda[r]/2.0^{(N+1)}$ (N is the number of times for rolling back to round k)
23: $T[r+1] = T[k] + \lambda[r+1]$
24: **end if**
25: **end if**
26: **end for**

gressiveness of pruning, and λ determines the increment of T at each pruning round. Pruning starts conservatively, with T initialized to 0, so that only completely useless filters that cannot capture any features are pruned. After each round of pruning, if the model accuracy loss is below than the target accuracy loss, it is considered "acceptable", and the algorithm increases the aggressiveness of pruning by incrementing T by λ. As pruning becomes increasingly aggressive, the accuracy eventually drops below the target accuracy at a certain round which is considered "unacceptable". When this happens, our algorithm rolls back the model weights and pruning threshold to the last acceptable round where the accuracy loss is within the target, and restarts the pruning from there but more conservatively—it increases the threshold more slowly by cutting the λ value by half. If this still does not lead to an acceptable round, the algorithm cuts λ by half again and restarts again. If after several trials, the accuracy loss is still not acceptable, the algorithm rolls back even further and restarts from an earlier round. The rationale behind this adaptive algorithm is that the aggressiveness of pruning should accelerate when the model is far from the pruning target, and decelerate when it is close to the target. Eventually, the changes of model size converges (when the changes of the number of parameters is within 0.1% for several rounds) and the algorithm terminates.

2.3 Layer-aware Threshold Adjustment

While adapting the global pruning threshold using the above discussed policies, our pruning method further considers the difference in each layer's contribution to model size and complexity and use differentiated layer-specific thresholds to prune the layers. As illustrated in Figure 1(b), in terms of the contribution to model size, the number of parameters of layer i can be estimated as $N^i = n^{(i-1)} \times k^i \times k^i \times n^i$; in terms of the contribution to computational complexity, the number of FLOPs of layer i can be estimated as $2 \times h^i \times w^i \times N^i$. Therefore, to effectively prune a model while maintaining its accuracy, we need to treat each layer differently at each round of the iterative pruning process based on its current contributions to the model size and complexity. Specifically, our adaptive pruning method calculates a weight for each layer based on its contribution and then uses this weight to adjust the current global threshold and derive a local threshold for the layer. If the goal is to reduce model size, the weight is calculated as each layer's number of parameters divided by the model's total number of parameters; if the goal is to reduce model computational complexity, the

Algorithm 3 Layer-aware Threshold Adjustment

1: **for** the layer i in a pruning round r **do**
2: Calculate the number of parameters of unpruned filters of the current layer and the whole model, respectively: $N^i[r] = n^{(i-1)}[r] \times k^i[r] \times k^i[r] \times n^i[r]$, $N^{Total}[r] = \sum_{i=1}^{i=M} n^{(i-1)}[r] \times k^i[r] \times k^i[r] \times n^i[r]$, where M is the number of convolution layers, and for other notations refer to Section 2.1.
3: Calculate the threshold $T^i[r]$ of the current layer: $N^i[r] = T[r] \times \frac{N^i[r]}{N^{Total}[r]}$.
4: For each filter F_j^i in F^i, calculate its attention: $F_{mean}(A_j^i) = \frac{1}{h^i \times w^i} \sum_{k=1}^{h^i} \sum_{l=1}^{w^i} \left|a_{k,l}^i\right|^p$.
5: Prune the filter if its attention is not larger than threshold $T^i[r]$, i.e., set $M_j^i = 0$ if $F_{mean}(A_j^i) \leqslant T^i[r]$; Otherwise, set $M_j^i = 1$
6: **end for**

Table 1: Comparing the proposed method with state-of-the-art structured pruning methods on the CIFAR-10 dataset. The baseline Top-1 accuracy of ResNet-56 and ResNet-50 on CIFAR-10 is 92.84%, 91.83% (5 runs), respectively. The numbers of the related works are cited from their papers.

Model	Target	Target Level	Method	Acc. ↓ (%)	Params. ↓ (%)	FLOPs. ↓ (%)
ResNet-56	Acc. ↓ (%)	0%	SCOP	0.06	56.30	56.00
			HRank	0.09	42.40	50.00
			PFF	0.03	42.60	43.60
			ILP	0.00	13.04	-
			PPR	-0.03	-	47.00
			Ours-1	**-0.33**	**79.11**	**-**
			Ours-2	**-0.08**	**-**	**70.13**
		1%	RFR	1.00	50.00	-
			ILP	1.00	41.18	-
			GAL	0.52	44.80	48.50
			Ours-1	**0.86**	**88.23**	**-**
			Ours-2	**0.77**	**-**	**81.19**
	Params. ↓ (%)	70%	DCP	-0.01	70.30	47.10
			GBN	0.03	66.70	70.30
			HRank	2.38	68.10	74.10
			Ours	**-0.6**	**71.57**	**-**
		50%	CP	1.00	-	50.00
			SFP	1.33	50.60	52.60
			FPGM	0.10	50.60	52.60
			Ours	**-1.08**	**53.3**	**-**
	FLOPs. ↓ (%)	75%	HRank	2.38	68.10	74.10
			DH	2.54	-	71.00
			PPR	1.17	-	71.00
			Ours	**0.3**	**-**	**71.44**
		55%	CP	1	-	50.00
			SFP	1.33	50.60	52.60
			FPGM	0.10	50.60	52.60
			AMC	0.90	-	50.00
			Ours	**-0.63**	**-**	**52.92**
ResNet-50	Params. ↓ (%)	60%	AMC	-0.02	60.00	-
			Ours	**-0.86**	**64.81**	**-**

weight is calculated as each layer's FLOPs divided by the model's total FLOPs. These layer-specific thresholds are then used to prune the layers in the current pruning round.

Assuming the goal is to reduce model size, the procedure in a pruning round r for the ith conv2d layer is shown in Algorithm 3 (the algorithm for reducing model FLOPs is similar). For pruning, we introduce a mask $M_j^i \epsilon \{0,1\}^{n^{i-1} \times k^i \times k^i}$ for each filter F_j^i. The effective weights of the convolution layer after pruning is the element-wise product of the filter and mask—$F_j^i \odot M_j^i$. For a filter, all values of this mask is either 0 or 1, thus either keeping or pruning the entire filter.

Table 2: Comparing the proposed method with state-of-the-art structured pruning methods on ResNet-50 with ImageNet dataset. The baseline Top-1 accuracy of ResNet-50 on ImageNet is 74.97% (5 runs). The result of ILP is from our implementation, and all the other numbers of the related works are cited from their papers.

Model	Targets	Targets Level	Method	Acc. ↓ (%)	Params. ↓ (%)	FLOPs ↓ (%)
ResNet-50	Acc loss ↓ (%)	0%	PFP-A	0.22	18.10	10.80
			Ours-1	**-0.12**	**24.55**	**-**
			Ours-2	**0.18**	**-**	**11.62**
		1%	SSS-41	0.68	0.78	15.06
			ILP	1.00	11.5	-
			Ours-1	**0.64**	**30.39**	**-**
			Ours-2	**0.83**	**-**	**32.26**
	Params. ↓ (%)	40%	Hrank	1.17	36.70	43.70
			SSS-26	4.30	38.82	43.04
			Ours	**1.09**	**37.17**	**-**
		30%	PFP-B	0.92	30.10	44.00
			Ours	**0.48**	**29.42**	**-**
	FLOPs. ↓ (%)	30%	SSS-32	1.94	27.06	31.08
			Ours	**0.57**	**-**	**31.88**

3 EVALUATION

3.1 EXPERIMENTAL SETUP

Models and Datasets. We evaluated our proposed method on representative image classification DNNs and datasets: 1) ResNet-56 and ResNet-50 (He et al. (2016)) on CIFAR-10 (Krizhevsky et al. (2009)); and 2) ResNet-50 (He et al. (2016)) on ImageNet (Deng et al. (2009)).

Implementation Details. We implemented our proposed method on PyTorch version 1.6.0. For ResNet models on CIFAR-10, the learning rate is set to 0.1 initially and decays with a rate of 0.1 at the epoch 91 and 136. Weight decay is set to 0.0002. For ResNet-50 on ImageNet, the learning rate increases to 0.256 in a warmup mechanism during the first 5 epochs, and decays with a factor of 0.1 at epochs 30, 60, 80 (Renda et al. (2020); Frankle et al. (2019)). Nesterov SGD optimizer is used with a momentum of 0.9 for all models. The simple data augmentation (random crop and random horizontal flip) is used for all training images. Learning rate rewinding is used for iterative pruning, which rewinds to the epoch at roughly 60% of the total training duration (see Appendix A.4 for an analysis of the effect of rewinding epochs on the accuracy of the pruned models). For adaptive pruning, T and λ are initialized to 0.0 and 0.005, respectively.

Evaluation Metrics. We evaluate a pruned network on the following three criteria: 1) *Accuracy:* Top-1 accuracy; 2) *Model Size:* total number of parameters. For pruned models, we consider only the unmasked weights; 3) *Compute Intensity:* the total number of floating point operations (FLOPs) required to perform model inference. For pruned models, we consider only the computation of the unmasked weights.

3.2 CIFAR-10 RESULTS

Table 1 lists the results from CIFAR-10. We compare our method with the state-of-the-art works, including **SCOP** (Tang et al. (2020)), **HRank** (Lin et al. (2020)), **ILP** (Renda et al. (2020)), **PPR** (Zhuang et al. (2020)), **RFR** (He et al. (2017)), **GAL** (Lin et al. (2019)), **DCP** (Zhuang et al. (2018)), **GBN** (You et al. (2019)), **CP** (He et al. (2017)), **SFP** (He et al. (2018a)), **FPGM** (He et al. (2019)), DeepHoyer (**DH**) (Yang et al. (2019)), and **AMC** (He et al. (2018b)). Note that a negative accuracy drop means that the pruned model achieves better accuracy than its unpruned baseline model. "Ours-1" and "Ours-2" denote the cases where the pruning objective is set to minimize model size and FLOPs, respectively, while meeting the accuracy target.

In all cases targeting accuracy, model size, and compute intensity, our method significantly outperforms the recent related works. For example, on ResNet-56, without accuracy drop, our method achieves the largest parameter reduction (79.11%), outperforming the related works by 22.81% to 66.07%, and the largest FLOPs reduction (70.13%), outperforming the related works by 14.13% to 26.53%. With 70% of parameter reduction, our method achieves the smallest accuracy loss (-0.6%),

outperforming the related works by 0.59% to 2.98%. Also, with 75% of FLOPs reduction, our method achieves the smallest accuracy loss (0.3%), outperforming the related works by 0.87% to 2.24%. Note that the proposed method also produces a pruned model that reaches 0.6% or 1.08% higher accuracy than the original model but with only 30% or 50%, respectively, of the original's parameters. Such small and accurate models are useful for many real-world applications.

The results validate that activation-based pruning is more effective than weight magnitude based pruning in finding unimportant filters, as discussed in Section 2.1. For example, on ResNet-56, with 1% of accuracy loss, the amount of parameter reduction achieved by our method is higher than that of ILP (Renda et al. (2020)) by 34.51%; and with 75% FLOPs reduction, the accuracy achieved by our method is higher than PPR (Zhuang et al. (2020)) by 0.87%.

The results confirm that our method can achieve better results that the state-of-the-art automatic pruning methods, AMC (He et al. (2018b)). For example, when targeting at about 55% of FLOPs reduction on ResNet-56, our method achieves 1.53% higher accuracy than AMC; and when targeting at about 60% of parameters reduction on ResNet-50, our method achieves 0.84% higher accuracy than AMC. Our activation-based adaptive pruning can directly target unimportant filters in the model, whereas AMC has to use reinforcement learning to explore the whole network.

To illustrate the effectiveness of our proposed adaptive pruning, in Figure 2, we show how the accuracy loss and parameter reduction change over the pruning rounds as the pruning threshold adapts according to Algorithm 2, in the experiment of pruning ResNet-56 on CIFAR-10, given the target of 1% accuracy loss. From Round 1 to Round 24, the accuracy loss of the pruned model is lower than the target accuracy loss (1%), so the algorithm increases the pruning aggressiveness gradually by increasing the threshold. At Round 25, the accuracy loss exceeds the target, so the algorithm rolls back the model weights and the pruning threshold to Round 24, and restart the pruning from there more conservatively. The above process repeats until after Round 39, the model size converges, and the algorithm terminates at reducing totally 88.23% of parameters.

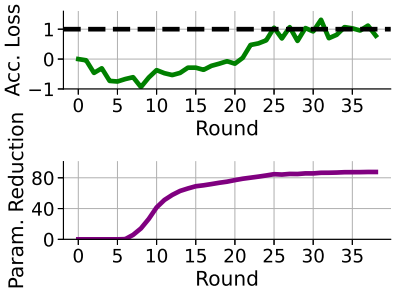


Figure 2: Illustration of the changes of the accuracy loss and parameter reduction over the pruning rounds as the pruning threshold adapts when pruning ResNet-56 with CIFAR-10, given the target of 1% Top-1 accuracy loss.

3.3 ImageNet Results

Table 2 shows the results of of ResNet-50 on ImageNet dataset, comparing our method with **PFB** (Liebenwein et al. (2019)), **HRank** (Lin et al. (2020)), **ILP** (Renda et al. (2020)), and Sparse Structure Selection (**SSS**) (Huang & Wang (2018)). "Ours-1" and "Ours-2" denote the cases where the pruning objective is set to minimize the number of parameters and FLOPs, respectively, while meeting the accuracy target. For the same level of accuracy loss, our method reduces significantly more parameters (6.45% to 29.61% higher than the related works) and more FLOPs (0.82% to 17.2% higher than the related works). For the same level of parameters or FLOPs reduction, our method achieves the smallest accuracy loss, lower than the related works by 0.08% to 3.21%.

4 Conclusions

This paper proposes an adaptive, activation-based, iterative structured pruning approach to automatically and efficiently generate small, accurate, and hardware-efficient models that can meet diverse user requirements. We show that activation-based attention value is a more precise indicator to identify unimportant filters to prune than the commonly used weight magnitude value. We also offer an effective way to perform structured pruning in an iterative pruning process, leveraging the LTH theory to find small and accurate sub-networks that are at the same time hardware efficient. Finally, we argue that automatic pruning is essential for this method to be useful in practice, and proposes an adaptive pruning method that can automatically meet user-specified objectives in terms of model accuracy, size, and inference speed but without user intervention. Our results confirm that the proposed method outperforms existing structured pruning approaches with a large margin.

REFERENCES

Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*, 2017.

H Cai, T Chen, W Zhang, Y Yu, and J Wang. Reinforcement learning for architecture search by network transformation. corr abs/1707.04873 (2017).

Moody T Chu, Robert E Funderlic, and Robert J Plemmons. Structured low rank approximation. *Linear algebra and its applications*, 366:157–172, 2003.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets, 2019.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018a.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4340–4349, 2019.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018b.

P. Hill, A. Jain, M. Hill, B. Zamirai, C. Hsu, M. A. Laurenzano, S. Mahlke, L. Tang, and J. Mars. Deftnn: Addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 786–799, 2017.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 304–320, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2016.

Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. *arXiv preprint arXiv:1911.07412*, 2019.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1529–1538, 2020.

Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2790–2799, 2019.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pp. 2498–2507. PMLR, 2017.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *arXiv preprint arXiv:2010.10732*, 2020.

Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

H Yang, W Wen, and H Deephoyer Li. Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*, 2019.

Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4133–4141, 2017.

Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1909.08174*, 2019.

Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, 2016.

Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. *Advances in Neural Information Processing Systems*, 33, 2020.

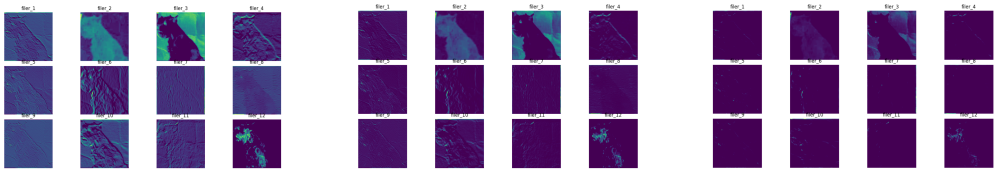
Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *arXiv preprint arXiv:1810.11809*, 2018.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A APPENDIX

A.1 THE EFFECT OF ATTENTIONS

Figure 3 shows the attention of each filter of the first convolution layer of ResNet-50 on ImageNet with different values of p (p=1, 2, 4). The setting where p is equal to 1 tends to be best, since iit promotes the effectiveness of the pruning by enabling the gap between the mean values of the useful filters and useless filters to be large.



(a) p=1 (b) p=2 (c) p=4

Figure 3: Attentions of each filter of the first convolution layer of ResNet-50 on ImageNet with different values of p ($p = 1, 2, 4$).

A.2 MEMORY-CONSTRAINED ADAPTIVE PRUNING

The Memory-constrained Adaptive Pruning Algorithm is shown in Algorithm 4.

Algorithm 4 Memory-constrained Adaptive Pruning

1: **Input:** Target Parameters Reduction $ParamTarget$
2: **Output:** A small pruned model with an acceptable model size
3: Initialize: $T = 0.0, \lambda = 0.005$.
4: **for** pruning round r ($r \geqslant 1$) **do**
5: Prune the model using $T[r]$ (Refer to Algorithm 3)
6: Train the pruned model, calculate its remaining number of parameters $Param[r]$
7: Calculate the parameters reduction: $ParamRed[r]$: $ParamRed[r] = Param[0] - Param[r]$
8: **if** $ParamRed[r] < ParamTarget$ **then**
9: **if** the changes of model size are within 0.1% for several rounds **then**
10: Terminate
11: **else**
12: $\lambda[r+1] = \lambda[r]$
13: $T[r+1] = T[r] + \lambda[r+1]$
14: **end if**
15: **else**
16: Find the last acceptable round k
17: **if** k has been used to roll back for several times **then**
18: Mark k as unacceptable
19: Go to Step 15
20: **else**
21: Roll back model weights to round k
22: $\lambda[r+1] = \lambda[r]/2.0^{(N+1)}$ (N is the number of times for rolling back to round k)
23: $T[r+1] = T[k] + \lambda[r+1]$
24: **end if**
25: **end if**
26: **end for**

A.3 FLOPS-CONSTRAINED ADAPTIVE PRUNING

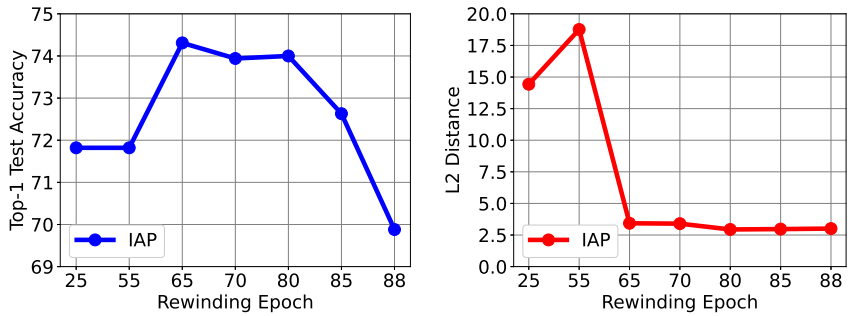
The FLOPs-constrained Adaptive Pruning Algorithm is shown in Algorithm 5.

Algorithm 5 FLOPs-constrained Adaptive Pruning

1: **Input:** Target FLOPs Reduction $FLOPsTarget$
2: **Output:** A small pruned model with an acceptable FLOPs
3: Initialize: $T = 0.0, \lambda = 0.005$.
4: **for** pruning round r ($r \geqslant 1$) **do**
5: Prune the model using $T[r]$ (Refer to Algorithm 3)
6: Train the pruned model, calculate its remaining FLOPs $FLOPs[r]$
7: Calculate the parameters reduction: $FLOPsRed[r]$: $FLOPsRed[r] = FLOPs[0] - FLOPs[r]$
8: **if** $FLOPsRed[r] < FLOPsTarget$ **then**
9: **if** the changes of FLOPs are within 0.1% for several rounds **then**
10: Terminate
11: **else**
12: $\lambda[r+1] = \lambda[r]$
13: $T[r+1] = T[r] + \lambda[r+1]$
14: **end if**
15: **else**
16: Find the last acceptable round k
17: **if** k has been used to roll back for several times **then**
18: Mark k as unacceptable
19: Go to Step 15
20: **else**
21: Roll back model weights to round k
22: $\lambda[r+1] = \lambda[r]/2.0^{(N+1)}$ (N is the number of times for rolling back to round k)
23: $T[r+1] = T[k] + \lambda[r+1]$
24: **end if**
25: **end if**
26: **end for**

A.4 The Effect of Rewinding Epoch

In order to understand how the rewinding impacts accuracy of the pruned models, we analyze *stability to pruning* as proposed by Frankle et al. (2019), which is defined as the L2 distance between the masked weights of the pruned network and the original network at the end of training. We validate the observations proposed by Frankle et al. (2019) that for deep networks, rewinding to very early stages is sub-optimal as the network has not learned considerably by then; and rewinding to very late training stages is also sub-optimal because there is not enough time to retrain. Specifically, Figure 4(a) shows the Top-1 test accuracy of the pruned ResNet-50 with 83.74% remaining parameters when learning rate is rewound to different epochs, and Figure 4(b) shows the stability values at the corresponding rewinding epochs. We observe that there is a region, 65 to 80 epochs, where the resulting accuracy is high. We find that L2 distance closely follows this pattern, showing large distance for early training epochs and small distance for later training epochs. Our findings show that rewinding to 75%-90% of training time leads to good accuracy.



(a) Top-1 Test Accuracy (b) Stability to Pruning

Figure 4: The effect of the rewinding epoch (x-axis) on (a) Top-1 test accuracy, and (b) pruning stability, for pruned ResNet-50, when 83.74% of parameters are remaining.