Automatic Attention Pruning: Improving and Automating Model Pruning using Attentions

Kaiqi Zhao
Arizona State University

Animesh Jain
Meta

Ming Zhao
Arizona State University

Abstract

Pruning is a promising approach to compress deep learning models in order to deploy them on resource-constrained edge devices. However, many existing pruning solutions are based on unstructured pruning, which yields models that cannot efficiently run on commodity hardware; and they often 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 Automatic Attention Pruning (AAP), an adaptive, attention-based, structured pruning approach to automatically generate small, accurate, and hardware-efficient models that meet user objectives. First, it proposes iterative structured pruning using activation-based attention 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 AAP substantially outperforms the state-of-the-art structured pruning works for a variety of model architectures. Our code is at: `https://github.com/kaiqi123/Automatic-Attention-Pruning.git`.

1 Introduction

Deep neural networks (DNNs) have substantial computational and memory requirements. As the use of deep learning grows rapidly on a wide variety of Internet of Things and devices, the mismatch between resource-hungry DNNs and resource-constrained devices also becomes increasingly severe. Pruning is a promising approach to identify and remove the parameters that do not contribute significantly to the accuracy of a DNN. Recent works based on the Lottery Ticket Hypothesis (LTH) have achieved great results in creating smaller and more accurate models (dubbed as "winning tickets") through iterative pruning with rewinding [Frankle and Carbin, 2018]. However, LTH has only been shown to work with unstructured pruning which, unfortunately, leads to models with low sparsity and difficult to accelerate on commodity hardware; e.g., directly applying NVIDIA cuSPARSE on unstructured pruned models can lead to a 60× slowdown compared to dense kernels on GPUs [Hill et al., 2017]. Moreover, most pruning methods require users to explore and adjust multiple hyper-parameters, which is time-consuming and often leads to sub-optimal results; e.g., with LTH-based iterative pruning, users need to determine how many parameters to prune in each pruning round.

We propose Automatic Attention Pruning (AAP), an *adaptive*, *attention-based*, *structured pruning* solution to automatically generate small, accurate, and hardware-efficient models that meet users' accuracy, size, and speed requirements. We improve the LTH-based iterative pruning framework by proposing two methods. First, we propose a novel attention pruning method to identify and remove unimportant filters. Specifically, we properly define an attention mapping function that takes the 2D activation feature map of a filter as input and outputs a 1D value used to indicate the importance of the filter. This approach is more effective than weight-value-based filter pruning [Renda et al., 2020, Zhuang et al., 2020, Wang et al., 2019] 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. Also, it is better than previous activation-based filter pruning methods [Lin et al., 2020, Liu et al., 2017] since the accuracy of its measurement does not depend on the amount of inputs.

Second, we propose an adaptive pruning method 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 with the acceptable inference speed; for memory-limited environ-

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

ments like embedded systems, we propose model-size-guaranteed pruning to achieve the best accuracy and fit the memory constraint; for accuracy-critical applications such as those on self-driving cars, we propose accuracy-guaranteed pruning to create the most resource-efficient model with the acceptable accuracy loss. Given the target, our method adaptively controls the pruning aggressiveness by adjusting the global threshold used to prune filters. Moreover, it recognizes the difference in each layer's contribution to the model's size and computational complexity and uses a layer-wise 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 a differentiated level of aggressiveness.

The proposed AAP outperforms the related works significantly in all cases targeting accuracy loss, parameters reduction, and FLOPs reduction for a variety of model architectures. For example, on ResNet-56 with CIFAR-10, without accuracy drop, AAP achieves the largest parameters 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%. On ResNet-50 with ImageNet, for the same level of parameters and FLOPs reduction, AAP achieves the smallest accuracy loss, lower than the related works by 0.08% to 2.61%; and for the same level of accuracy loss, AAP 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).

In summary, our main contributions are: 1) a novel iterative, structured pruning approach for finding the "winning ticket" models that are hardware efficient; 2) a new attention-based mechanism for accurately identifying unimportant filters for pruning, which is much more effective than existing methods; and 3) an adaptive pruning method that can automatically optimize the pruning process according to diverse real-world scenarios.

2 Background and Related Works

Unstructured Pruning vs. Structured Pruning. Unstructured pruning (e.g., [LeCun et al., 1990, Han et al., 2015a, Molchanov et al., 2017]) prunes individual elements in the weight tensors of a model. It has less impact on model accuracy, compared to structured pruning, because it is finer-grained, but unstructured pruned models are hard to accelerate on commodity hardware. Structured pruning is a coarser-grained approach that prunes entire regular regions of the weight tensors of a model. 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. 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 Pruning vs. Iterative Pruning. One-shot pruning prunes a pre-trained model and then retrains it once, whereas iterative pruning prunes and retrains the model in multiple rounds. Iterative pruning generally achieves much better performance than one-shot pruning because multiple retraining phases help recover the accuracy lost during pruning. In particular, recent works based on the Lottery Ticket Hypothesis (LTH) have achieved great results in creating smaller and more accurate models through iterative pruning with rewinding [Frankle and Carbin, 2018]. LTH posits that a dense network has a sub-network, termed as a "winning ticket", which can achieve an accuracy comparable to the original network. However, existing LTH-based works consider only unstructured pruning, e.g., Iterative Magnitude Pruning (IMP) [Frankle and Carbin, 2018, Frankle et al., 2019] and Synflow [Tanaka et al., 2020], which, as discussed above, are hardware-inefficient.

Automatic Pruning. For pruning to be useful in practice, it is important to automatically meet the pruning objectives for diverse machine learning applications and devices. Most pruning methods require users to explore and tune multiple hyper-parameters, e.g., with LTH-based pruning, users need to determine how many parameters to prune in each round. Manual tuning is time-consuming and often leads to sub-optimal results. Some works use learning-based methods to find smaller models in a Neural Architecture Search (NAS) approach: AMC [He et al., 2018b], NAS [Zoph et al., 2018], NT [Cai et al.,], and N2N [Ashok et al., 2017] use reinforcement learning, and GAL [Lin et al., 2019] uses adversarial learning. But these methods have to explore a large search space of all available layer-wise sparsity, which is time consuming when neural networks are large and datasets are complex.

Therefore, there is a great need for an automatic, iterative, structured pruning solution that can automatically and efficiently generate small, accurate, and hardware-efficient models. The challenges are three-fold.

First, *how to effectively identify the insignificant parameters in a model to prune?* Existing works have explored different mechanisms, e.g., L2-Norm in Soft Filter Pruning (SFP) [He et al., 2018a], Soft Channel Pruning (SCP) [Kang and Han, 2020] and EagleEye [Li et al., 2020a], geometric median in FPGM [He et al., 2019], Hessian in EigenDamage [Wang et al., 2019], Empirical Sensitivity in Provable Filter Pruning (PFP) [Liebenwein et al., 2019], adversarial knockoff features in SCOP [Tang et al., 2020], polarization regularizer in Neuron-level Structured Pruning (NSP) [Zhuang et al., 2020], LASSO regression in Channel Pruning (CP) [He et al., 2017], and other information considering the relationship between

neighboring layers (Gate Batch Normalization (GBN) [You et al., 2019], Sparse Structure Selection (SSS) [Huang and Wang, 2018], Hinge [Li et al., 2020b], Pruning From Scratch (PFS) [Wang et al., 2020] and Stripe-Wise Pruning (SWP) [Meng et al., 2020]). In comparison, *activation-based attention*, proposed in this paper, can more effectively capture the importance of filters, and pruning based on attention values can produce much better models, as quantitatively shown in our evaluation (Section 4).

Second, *how to design an effective iterative pruning process to recover the accuracy loss caused by structured pruning?* LTH-based iterative pruning is a promising approach, but it has only been shown to work with unstructured pruning such as IMP. Its counterpart in structured pruning—Iterative L1-norm-based pruning (ILP) [Renda et al., 2020], which removes filters based on their L1-norm values, cannot effectively prune a model while maintaining its accuracy. For example, ILP can prune ResNet-50 by at most 11.5% of parameters when the maximum accuracy loss is limited to 1% on ImageNet. So directly applying iterative pruning with existing weight-magnitude-based structured pruning methods does not produce accurate pruned models. This paper proposes a novel *LTH-based iterative, structured pruning* solution using attentions, and it significantly outperforms ILP and other related structured pruning works that involve an iterative process (GDP [Guo et al., 2021], ACTD [Wang et al., 2021], Quantization and Pruning (QP) [Paupamah et al., 2020], IMP-Refill and IMP-Regroup [Chen et al., 2022]).

The third challenge is *how to automate the pruning process so it does not require any human intervention?* The existing structured pruning works all require difficult hand-tuning of many hyper-parameters, e.g., DCP [Zhuang et al., 2018] and MDP [Guo et al., 2020] need multiple hyper-parameters to balance the original task-specific loss and the additional pruning loss; VCNNP [Zhao et al., 2019] requires careful settings of τ and θ to decide which filters to prune; DMC [Gao et al., 2020] and DeepHoyer [Yang et al., 2019] require parameters to decide the regularization strength with different settings for different datasets and models. To address this challenge, this paper proposes a fully automated pruning solution that can automatically generate pruned models that meet users' diverse model accuracy, size, and speed requirements.

3 Methodology

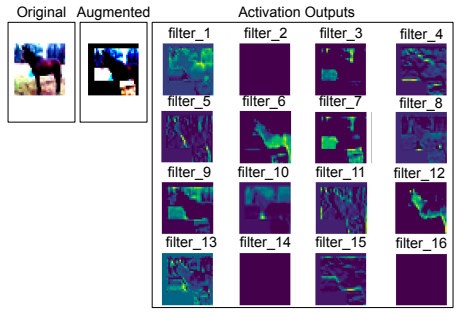
Algorithm 1 lists the proposed adaptive structured pruning for AAP. We improve LTH-based iterative pruning by proposing activation-based attention pruning (Lines 7 and 8) and adaptive pruning policies (Lines 11–14), to automatically and efficiently generate a pruned model that meets the user's different objectives. To represent pruning of weights, we use a mask $M^r \epsilon \{0,1\}^{|W^r|}$ for each weight tensor W_t^r, where r is the pruning round number and t is the training epoch. Therefore, the pruned network at the end of training epoch E is represented by the element-wise product $M^r \odot W_E^r$. Lines 3–5 are to train the original model to completion while saving the weights at epoch k. Lines 6–15 represent a pruning round. Lines 7 and 8 prune the model (discussed in Section 3.1). Lines 9 (optional) and 10 perform rewinding. Line 11 retrains the pruned model for the remaining $E-k$ epochs. Line 12 evaluates the pruned model according to the pruning target. If the target is not met, Line 13 resets the weights to an earlier round. Line 14 calculates the threshold for the next pruning round following the adaptive pruning policy (discussed in Section 3.2)

Algorithm 1 Adaptive Iterative Structured Pruning

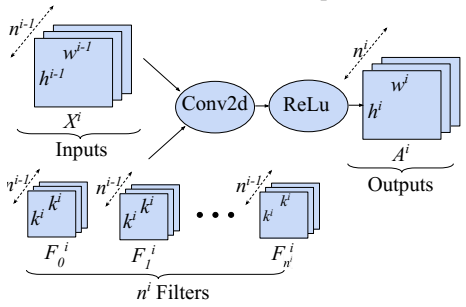
1: **Input:** An uncompressed network, and the pruning target
2: **Output:** A pruned network that meets the target
3: [Initialize] Initialize a network $f(x; M^0 \odot W_0^0)$ with the initial mask $M^0 = \{0,1\}^{|W_0^0|}$
4: [Save weights] Train the network for k epochs, yielding network $f(x; M^0 \odot W_k^0)$, and save weights W_k^0
5: [Train to converge] Train the network for $E-k$ epochs to converge, producing network $f(x; M^0 \odot W_E^0)$
6: **for** pruning round r ($r \geqslant 1$) **do**
7: [Calculate attention] Calculate the attention value of each filter using the attention mapping function $F(\cdot)$
8: [Prune] From W_E^{r-1}, prune filters with an attention value less than $T[r]$, producing a mask M^r and a network $f(x; M^r \odot W_E^{r-1})$
9: [Rewind Weights] Reset the remaining filters to W_k^0 at epoch k, producing network $f(x; M^r \odot W_k^{r-1})$
10: [Rewind Learning Rate] Reset the learning rate schedule to its state from epoch k
11: [Retrain] Retrain the unpruned filters for $E-k$ epoch to converge, yielding network $f(x; M^r \odot W_E^r)$
12: [Evaluate] Evaluate the retrained network $f(x; M^r \odot W_E^r)$ according to the target
13: [Reset Weights] If the target is not met, reset the weights to an earlier round
14: [Adapt Threshold] Calculate the next threshold $T[r+1]$
15: **end for**

3.1 Attention-based Filter Pruning

First, we propose that, compared with the *weight values* of a filter, its *activation values* are more effective indicators of 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. On the other hand, 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. We present a visual motivation in Figure 1a. The figure shows



(a) Filter activation outputs.



(b) Convolution layer tensors.

Figure 1: (a) Activation outputs of 16 filters, and (b) input and output tensors of a convolution layer.

the activation outputs 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 filters. Therefore, from visual inspection, removing filters with weak activation patterns is likely to have a low impact on the final accuracy of the pruned model.

Thus the key problem is to design a proper function that can reflect the useful information of the activation feature maps of each filter. Previous activation-based filter pruning methods address this problem using different forms of activations: NN Slimming [Liu et al., 2017] measures Average Percentage of Zeros (APoZ) of the activations; HRank [Lin et al., 2020] conductes a Singular Value Decomposition (SVD) for activations; AP+Coreset [Dubey et al., 2018] computes the mean value of activations; Provable Filter Pruning (PFP) [Liebenwein et al., 2019] computes the sensitivity of activations. However, these data-driven approaches require a large amount of inputs (e.g., 50,000 of ImageNet images for NN Slimming and all training samples for AP+Coreset) to achieve a reasonably accurate prediction and leads to data-dependent compressed models. Instead, we aim to design a function that is data-independent and robust to inputs: no matter what the input image is, the filters that can extract useful image features should always be maintained.

Activation-based attention is a good indicator of neurons regarding their ability to capture features [Zagoruyko and Komodakis, 2016]. Attention has been proven to be useful in various tasks, including neural machine translation [Bahdanau et al., 2014], object localization [Oquab et al., 2015], and knowledge transfer for image classification [Zagoruyko and Komodakis, 2016]. Also, motivated by human attention mechanism theories, attention maps can be obtained by computing a Jacobian of network outputs regarding the inputs [Simonyan et al., 2013], guided backpropagation [Springenberg et al., 2014], or converting the linear classification layer into a convolutional layer [Zhou et al., 2016]. However, the effectiveness of using attention as a mechanism for model compression is currently unexplored, and it imposes new challenges, e.g., the attention mapping function defined in Attention Transfer [Zagoruyko and Komodakis, 2016] outputs a flattened 2D matrix representing the ability to capture features of the whole convolution layer, not individual filters in that layer, and cannot be used to prune individual filters.

We address the challenge of effectively identifying insignificant filters in a network with novel designs for the attention mapping function. We start with some notations shown in Figure 1b. For the ith 2D convolution (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 proposed 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 mapping functions, where $p \geqslant 1$ and $a_{k,l}^i$ denotes every element of A_j^i:

1. **Attention Mean** (mean of the activation values)
   $F_{mean}(A_j^i) = \frac{1}{h^i \times w^i} \sum_{k=1}^{h^i} \sum_{l=1}^{w^i} \left| a_{k,l}^i \right|^p$;
2. **Attention Max** (max of the activation values)
   $F_{max}(A_j^i) = max_{l=1,h^i \times w^i} \left| a_{k,l}^i \right|^p$;
3. **Attention Sum** (sum of the activation values)
   $F_{sum}(A_j^i) = \sum_{k=1}^{h^i} \sum_{l=1}^{w^i} \left| a_{k,l}^i \right|^p$.

From these three, we choose Attention Mean, $F_{mean}(A_j^i)$ with p equal to 1 as the indicator to identify and prune unimportant filters. Also, in contrast to the related works, our proposed attention mapping function is robust to the

Algorithm 2 Accuracy-guaranteed Adaptive Pruning

1: **Input:** A converged uncompressed network and the target accuracy loss $AccLossTarget$
2: **Output:** The smallest model meeting the accuracy target
3: Initialize: $T = 0.0, \lambda = 0.01$.
4: **for** pruning round r ($r \geqslant 1$) **do**
5: Prune the model using $T[r]$ (Refer to Lines 7 and 8 in Algorithm 1)
6: Rewind weights and learning rate (Refer to Lines 9 and 10 in Algorithm 1)
7: Train the pruned model, and evaluate its accuracy $Acc[r]$ (Refer to Lines 11 and 12 in Algorithm 1)
8: Calculate the accuracy loss $AccLoss[r]$: $AccLoss[r] = Acc[0] - Acc[r]$
9: **if** $AccLoss[r] < AccLossTarget$ **then**
10: **if** the changes of model size are within 0.1% for several rounds **then**
11: Terminate
12: **else**
13: $\lambda[r+1] = \lambda[r]$
14: $T[r+1] = T[r] + \lambda[r+1]$
15: **end if**
16: **else**
17: Find the last acceptable round k
18: **if** k has been used to roll back for several times **then**
19: Mark k as unacceptable
20: Go to Step 17
21: **else**
22: Roll back model weights to round k
23: $\lambda[r+1] = \lambda[r]/2.0^{(C+1)}$ (C is the number of times for rolling back to round k)
24: $T[r+1] = T[k] + \lambda[r+1]$
25: **end if**
26: **end if**
27: **end for**

inputs, including real data or arbitrary random vectors; the attention values are calculated by only one batch of randomly chosen training data. See Section 4.4 for ablation studies on the choices of attention functions and the effect of data for evaluating attention values.

To the best of our knowledge, we are the first to study the effectiveness of using attention theories for model compression tasks such as structured pruning and solve the challenges by designing novel attention mapping functions that are effective for filter pruning.

3.2 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 requirements, which are 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 requirements; 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 pruning, we propose three adaptive pruning policies to provide 1) Accuracy-guaranteed pruning which produces the most resource-efficient model with the acceptable accuracy loss; 2) Memory-constrained pruning which generates the most accurate model within a given memory footprint; and 3) FLOPs-constrained pruning which creates the most accurate model within a given computational intensity. Specifically, our 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) as well as multi-objective optimization can also be readily supported (See Section 4.5). We take Accuracy-guaranteed Adaptive Pruning, described in Algorithm 2, as an example to show the procedure of adaptive pruning. Algorithm 2 is a specific version of Algorithm 1 with the acceptable accuracy loss set as the pruning target. Other versions with memory and FLOPs targets are included in Appendix A.1.

In the algorithm, T controls the aggressiveness 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, if the model accuracy loss is below the target accuracy loss, it is considered "acceptable", and the algorithm increases the aggressiveness of pruning by incrementing T by λ, with λ initialized to 0.01. As pruning becomes increasingly aggressive, the accuracy eventually drops below the target in 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.

Note that the value of λ continuously decreases as the algorithm gets close to the target, which guarantees the convergence of the pruned model: when λ becomes sufficiently

small, T does not grow much anymore, which means no more filters are pruned and the model converges. See Section 4.4 for an example. The algorithm terminates when the changes of model size is within 0.1% for several rounds.

The proposed algorithms can automatically—the only two parameters T and λ are automatically tuned—generate pruned models that meet diverse user requirements in model accuracy, size, and speed. Moreover, by making LTH-based iterative pruning adaptive, the algorithms can automatically find the best models that meet user requirements. Compared to related works which require time-consuming manual tuning of pruning parameters, AAP is the first to achieve these goals which are crucial to the practical use of model pruning for diverse real-world scenarios.

3.3 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 uses differentiated layer-specific thresholds to prune the layers. As shown in Figure 1b, 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 $F^i = 2 \times h^i \times w^i \times N^i$. A layer that contributes more to the model's size or FLOPs is more likely to have redundant filters to prune without affecting the model's accuracy.

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 remaining parameters $N^i[r]$ divided by the model's total number of remaining parameters $N^{Total}[r]$: $w^i[r] = \frac{N^i[r]}{N^{Total}[r]}$, where r is the pruning round. If the goal is to reduce model computational complexity, the weight is calculated as each layer's remaining FLOPs $F^i[r]$ divided by the model's total remaining FLOPs $F^{Total}[r]$: $w^i[r] = \frac{F^i[r]}{F^{Total}[r]}$. Then the threshold of layer i is calculated as: $T^i[r] = T[r] \times w^i[r]$. These layer-specific thresholds are then used to prune the layers in the current pruning round; they replace the global threshold $T[r]$ used to prune filters in Line 8 of Algorithm 1.

Compared to the related works which often use a single threshold to prune parameters for the entire network [Han et al., 2015b, Zhao et al., 2019], AAP's layer-specific thresholds allow it to generate better pruned models, and these thresholds are also fully automatically tuned.

Table 1: Implementation details

Dataset	Model	Learning Rate Schdeluder		Training Epochs	Weight decay
		Initial LR	Decay Epochs		
MNIST	LeNet-5	0.1	N/A	100	0.0
	LeNet-300-100	0.0012	N/A	100	0.0
CIFAR-10	ResNet-56	0.1	[91, 136]	182	2.00E-04
	ResNet-50	0.1	[91, 136]	182	2.00E-04
	ShuffleNet	0.1	[60, 120, 160]	200	4.00E-05
	MobileNet-V2	0.1	[150, 225]	300	4.00E-05
	VGG-16	0.05	[150, 180, 210]	240	5.00E-04
	VGG-19	0.05	[150, 180, 210]	240	5.00E-04
	LeNet-5	0.0002	N/A	24	1.00E-04
Tiny-ImageNet	ResNet-101	0.1	[150, 225]	300	2.00E-04
	VGG-19	0.1	[150, 225]	300	2.00E-04
ImageNet	ResNet-50	0.256	[30, 60, 80]	90	1.00E-04

4 Evaluation

We did an extensive evaluation on diverse models (ResNet, VGG, MobileNet, LeNet and ShuffleNet) and datasets (MNIST [Xiao et al., 2017], CIFAR-10 [Krizhevsky et al., 2009], Tiny-ImageNet [Le and Yang, 2015] and ImageNet [Deng et al., 2009]), and provided a thorough comparison to SOTA works (discussed in Section 2 and 3). For the proposed AAP, we consider different pruning policies: 1) meet accuracy target while minimizing the number of model parameters (*AAP-P*) or FLOPs (*AAP-F*); 2) meet parameter reduction target while minimizing accuracy loss (*AAP*); and 3) meet FLOP reduction target while minimizing accuracy loss (*AAP*). Multi-objective, multi-constraint pruning with AAP is also considered (Section 4.5).

Implementation Details We implemented AAP on PyTorch version 1.6.0 and conducted experiments on four Nvidia RTX 2080 GPUs. The implementation details are present in Table 1. The learning rate decays with a factor of 0.1 at decay epochs. Nesterov SGD optimizer is used with a momentum of 0.9. The batch size is set to 256, 128, 256, and 64 for the models on MNIST, CIFAR-10, Tiny-ImageNet, and ImageNet, respectively. Simple data augmentation (random crop and random horizontal flip) is used for all training images.

4.1 Results on CIFAR-10

In all cases targeting accuracy, model size, and compute intensity, the proposed method AAP significantly outperforms the recent related works. Table 2 shows the results from the widely used ResNet models on CIFAR-10. For example, for ResNet-56, without accuracy drop, AAP achieves the largest parameters 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 parameters reduction, AAP achieves the smallest accuracy

Table 2: Results from ResNet-56 and ResNet-50 on CIFAR-10. For the Acc. ↓ (%) column, a negative value means an increase in accuracy. The baseline Top-1 accuracy of ResNet-56 and ResNet-50 is 92.84% and 91.83%, respectively.

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
			SWP	0.03	42.60	43.60
			ILP	0.00	13.04	-
			NSP	-0.03	-	47.00
			EagleEye	-1.40	-	50.41
			AAP-P	-0.33	**79.11**	**56.97**
			AAP-F	-0.08	**65.78**	**70.13**
		1%	CP	1.00	50.00	-
			ILP	1.00	41.18	-
			GAL	0.52	44.80	48.50
			AAP-P	0.86	**88.23**	**70.09**
			AAP-F	0.77	**78.69**	**81.19**
	Params. ↓ (%)	70%	DCP	-0.01	70.30	-
			GBN	0.03	66.70	-
			HRank	2.38	68.10	-
			AAP	**-0.60**	71.57	-
		50%	SFP	1.33	50.60	-
			FPGM	0.10	50.60	-
			AAP	**-1.06**	53.89	-
	FLOPs. ↓ (%)	75%	HRank	2.38	-	74.10
			AAP	**0.97**	-	75.97
		70%	DeepHoyer	2.54	-	71.00
			NSP	1.17	-	71.00
			AAP	**0.30**	-	71.44
		55%	CP	1.00	-	50.00
			SFP	1.33	-	52.60
			FPGM	0.10	-	52.60
			AMC	0.90	-	50.00
			SCP	0.46	-	51.50
			AAP	**-0.63**	-	52.92
ResNet-50	Params. ↓ (%)	60%	AMC	-0.02	60.00	-
			AAP	**-0.86**	64.81	-

Table 3: Results from VGG, LetNet, MobileNet, and ShuffleNet models on CIFAR-10. For the Acc. ↓ (%) column, a negative value means an increase in accuracy. The baseline Top-1 accuracy of VGG-16, VGG-19, MobileNet-V2, ShuffleNet, and LeNet-5 are 93.64%, 93.90%, 94.46%, 93.28%, and 69.67%, respectively.

Model	Target	Target Level	Method	Acc. ↓ (%)	Params. ↓ (%)	FLOPs. ↓ (%)
VGG-16	Acc. ↓ (%)	0%	PFS	-0.19	-	50.00
			VCNNP	0.07	-	60.90
			Hinge	0.43	-	39.07
			HRank	0.53	-	53.60
			AAP-F	-0.16	**72.85**	**61.17**
	Params. ↓ (%)	70%	IMP-Refill	0.10	67.00	-
			IMP-Refill+	0.63	70.00	-
			IMP-Regroup	0.10	69.00	-
			AAP	**-0.27**	70.04	-
		80%	IMP-Refill	0.55	80.00	-
			IMP-Refill+	-	80.00	-
			IMP-Regroup	-0.05	80.00	-
			AAP	**-0.09**	81.21	-
VGG-19	Acc. ↓ (%)	0%	EigenDamage	0.19	78.18	37.13
			NN Slimming	1.33	80.07	42.65
			PFS	-0.31	-	52.00
			AAP-P	-0.26	**85.99**	**56.22**
			AAP-F	-0.03	**87.63**	**61.31**
	FLOPs. ↓ (%)	85%	EigenDamage	1.88	-	86.51
			AAP	**1.81**	-	89.02
MobileNet -V2	Acc. ↓ (%)	0%	GDP	-0.26	-	46.22
			MDP	-0.12	-	28.71
			SCOP	0.24	-	40.30
			DMC	-0.26	-	40.00
			AAP-P	-0.28	**79.68**	**55.67**
			AAP-F	-0.26	**76.79**	**58.99**
ShuffleNet	Acc. ↓ (%)	0%	QP	0.31	28.57	-
			AAP-P	0.19	**50.87**	**26.67**
LeNet-5	Params. ↓ (%)	90%	ILP	10.24	89.60	-
			AAP	**1.85**	90.38	-

loss (-0.6%), outperforming the related works by 0.59% to 2.98%. Also, with 70% of FLOPs reduction, AAP achieves the smallest accuracy loss (0.3%), outperforming the related works by 0.87% to 2.24%. Note that the proposed AAP also produces a pruned model that reaches 0.6% or 1.06% higher accuracy than the original model but with only 28.43% or 46.11%, respectively, of the original parameters. Such small and accurate models are useful for many real-world applications.

Table 3 shows the results from other model architectures. For example, on VGG-16, without any accuracy drop, AAP achieves the largest FLOPs reduction (61.17%), outperforming the related works by 0.27% to 22.1%. AAP can also effectively compress models (MobileNet, ShuffleNet) that are already designed to be compact. For example, on MobileNet-V2, without an accuracy drop, AAP achieves the largest FLOPs reduction (58.99%), speeding up this already lightweight model by more than half and outperforming the related works by 12.77% to 30.28%.

4.2 Results on ImageNet and Tiny-ImageNet

Table 4 shows that AAP can also significantly outperform the SOTA methods for various models trained on ImageNet and Tiny-ImageNet. For example, on ResNet-50 with ImageNet, for the same level of accuracy loss, AAP 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, AAP achieves the smallest accuracy loss, lower than the related works by 0.08% to 2.61%. On VGG-19 with Tiny-ImageNet, for the same level of parameters reduction, AAP achieves significantly lower accuracy loss than NN Slimming [Liu et al., 2017] by 11%. On ResNet-101 with Tiny-ImageNet, without accuracy loss, AAP achieves the highest parameters reduction (92.72%), outperforming the related works by 17.72% to 47.72%.

4.3 Results on MNIST

Table 5 shows that AAP can also significantly outperform the related works for LeNet-5 and LetNet-300-100 trained on MNIST. On LeNet-5, with 99% of parameters reduction, AAP achieves the smallest accuracy loss (-0.01%), outperforming the related works by 0.01% to 0.36%. Note that ADMM-NN-S [Ma et al., 2021] applies quantization after pruning, and AAP can also be further improved using quantization. Even without quantization, with the same level of parameters reduction, AAP achieves a lower accuracy loss than ADMM-NN-S by 0.21%. On LeNet-300-100, with 90% of parameters reduction, AAP achieves a lower accuracy loss than PFP [Liebenwein et al., 2019] by 0.03%.

Table 4: Results from ResNet-50 on ImageNet, and VGG-19 and ResNet-101 on Tiny-ImageNet. For the Acc. ↓ (%) column, a negative value means an increase in accuracy. The baseline Top-1 accuracy of ResNet-50 (ImageNet), VGG-19 (Tiny-ImageNet), and ResNet-101 (Tiny-ImageNet) are 75.06%, 59.64%, and 52.93%, respectively.

Model	Target	Target Level	Method	Acc. ↓ (%)	Params. ↓ (%)	FLOPs. ↓ (%)
ResNet-50	Acc. ↓ (%)	0%	PFP-A	0.22	18.10	10.80
			AAP-P	-0.12	**24.55**	**11.26**
			AAP-F	0.18	**24.09**	**11.62**
		1%	SSS-41	0.68	0.78	15.06
			ILP	1.00	11.50	-
			AAP-P	0.64	**30.39**	**22.70**
			AAP-F	0.83	**29.26**	**32.26**
	Params. ↓ (%)	40%	SSS-26	4.30	38.82	-
			Hrank	1.17	36.70	-
			AAP	**1.69**	40.85	-
		30%	PFP-B	0.92	30.10	-
			AAP	**0.80**	31.19	-
	FLOPs. ↓ (%)	30%	SSS-32	1.94	-	31.08
			AAP	**0.57**	-	31.88
VGG-19	Acc. ↓ (%)	3%	EigenDamage	3.36	61.87	66.21
			AAP-P	2.75	**73.69**	**79.87**
			AAP-F	2.61	**72.58**	**78.97**
	Params. ↓ (%)	60%	NN Slimming	10.66	60.14	-
			AAP	**-0.34**	60.21	-
ResNet-101	Acc. ↓ (%)	0%	NN Slimming	1.36	75.00	75.00
			GAL	0.50	45.00	76.00
			DHP	0.01	50.00	75.00
			ACTD	-0.44	51.00	75.00
			AAP-P	-0.57	**92.72**	**76.58**

Table 5: Results from LetNet-5 and LetNet-300-100 on MNIST. For the Acc. ↓ (%) column, a negative value means an increase in accuracy. The baseline Top-1 accuracy of LetNet-5 and LeNet-300-100 are 99.11% and 97.34%, respectively.

Model	Target	Target Level	Method	Acc. ↓ (%)	Params. ↓ (%)
LeNet-300-100	Params. ↓ (%)	90%	PFP	0.41	84.32
			AAP	**0.38**	89.94
LeNet-5	Params. ↓ (%)	99%	PFP	0.35	92.37
			AP+Coreset-K	0.00	99.39
			AP+Coreset-S	0.01	99.48
			AP+Coreset-A	0.01	99.48
			ADMM-NN-S	0.20	98.86
			AAP	**-0.01**	99.34

4.4 Ablation Study

The effect of attention-based iterative pruning. We fix the percentage of filters that the pruning process removes in each round, which is non-adaptive iterative pruning, and compare using our proposed attention values (termed IAP) with 1) using the conventional weight values, e.g. the L1-Norm of filters (as in Iterative L1-Norm Pruning (ILP) [Renda et al., 2020]), and 2) using the L1-Norm of filters divided by the filter size (termed ILP-Mean) to choose which filters to remove. Figure 2a shows the Top-1 accuracy of VGG-16 with the parameters reduction of 9.92%, 57.73% and 69.56% pruned by IAP, ILP, and ILP-Mean on CIFAR-10. IAP leads to a higher accuracy than the others by 0.1% to 0.58%.

The effect of attention mapping functions. Using VGG-16 on CIFAR-10 as an example, first, we analyze different types of attention mapping functions (discussed in Section 3.1), i.e., Attention Mean ($p = 1$), Attention Sum ($p = 1$), and Attention Max ($p = 1$), when used in one-shot attention-based pruning. With a parameter reduction of 57.73%, Attention Mean leads to the lowest top-1 accuracy loss, lower than Attention Sum and Attention Max by 0.25% and 0.13%, respectively. Then, we analyze Attention Mean with different values of p. When p is set to 1, it leads to the lowest accuracy loss: 0.38% when $p = 1$ vs 0.42% when $p = 2$ and 0.47% when $p = 4$. This confirms our choice of Attention Mean with $p = 1$ in our evaluation.

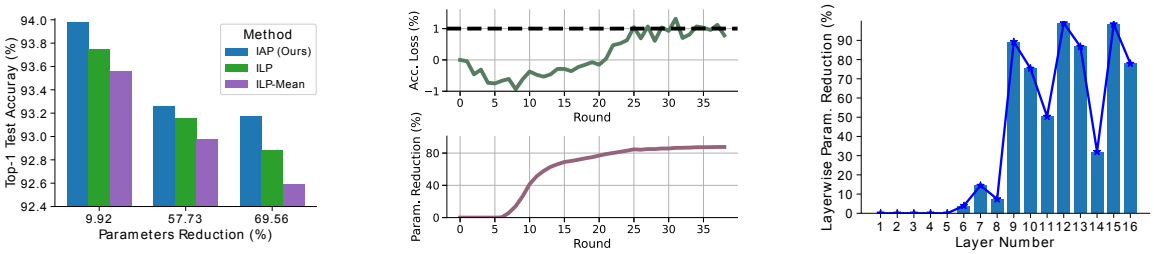
The effect of adaptive pruning. Using ResNet-56 on CIFAR-10 as an example, we show in Figure 2b how the adaptation of the pruning threshold (following Algorithm 2) affects accuracy loss and parameters reduction over the pruning rounds. From Round 1 to Round 24, the accuracy loss of the pruned model is 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 back to Round 24, and restarts the pruning from there more conservatively. The above process repeats until after Round 39, the model size converges, and the algorithm terminates at reducing 88.23% of parameters.

The effect of Layer-aware Threshold Adjustment. AAP considers the importance of each layer by adjusting differentiated layer-specific thresholds. Figure 2c shows the layer-wise sparsity of a pruned VGG-19 with a total parameters reduction of 85.99% on CIFAR-10. More filters are pruned from higher layers than lower layers.

The effect of inputs for evaluating attention values. The importance of each filter is evaluated by its attention value, which is calculated by one batch of randomly chosen training data after each pruning round (Line 7 in Algorithm 1). The attention values are *not sensitive* to the inputs because the model is converged from the training phase. As an example, Figure 3 shows the attention value of each filter of 4 convolution layers of ResNet-50 on ImageNet, given eight different batches of inputs, including randomly chosen real images and arbitrary random vectors. The attention value of each filter is consistent across different batches.

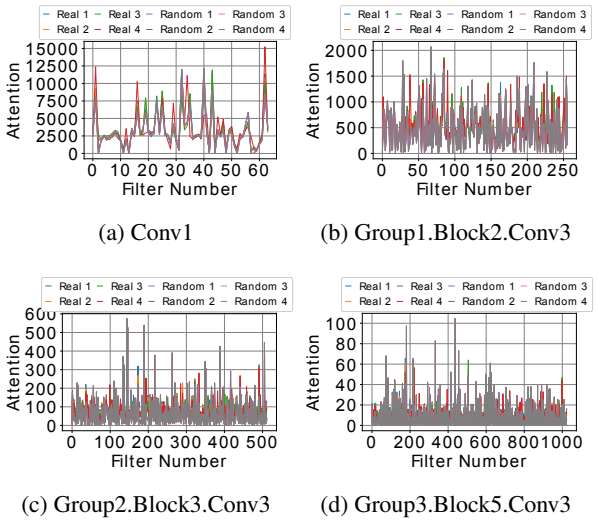
4.5 Discussions

Comparison to related works. The above results validate that 1) the proposed *activation-based attention pruning* is more effective than weight-magnitude-based pruning (e.g., EigenDamage [Wang et al., 2019], ILP [Renda et al., 2020], NSP [Zhuang et al., 2020]) in finding unimportant filters; 2) the proposed *activation-based attention pruning* is better than other activation-based pruning techniques that are based on different forms of activation feature maps, such as NN Slimming [Liu et al., 2017], HRank [Lin et al., 2020], AP+Coreset [Dubey et al., 2018]



(a) Effect of Attention Pruning (b) Effect of Adaptive Pruning (c) Layer-wise Sparsity

Figure 2: (a) Top-1 accuracy of VGG-16 iteratively pruned by the proposed attention pruning vs. L1-Norm based pruning on CIFAR-10. (b) Accuracy loss and parameter reduction over the pruning rounds as the pruning threshold is adapted following Algorithm 2. (c) Layer-wise sparsity of a pruned VGG-19 on CIFAR-10.



(a) Conv1 (b) Group1.Block2.Conv3

(c) Group2.Block3.Conv3 (d) Group3.Block5.Conv3

Figure 3: Attention values of filters from 4 convolution layers of ResNet-50 on ImageNet, given 8 different batches of inputs, including randomly chosen real images and arbitrary random vectors.

and PFP [Liebenwein et al., 2019]; and 3) the proposed *adaptive pruning* can achieve better results than other automatic pruning methods (e.g., AMC [He et al., 2018b] and GAL [Lin et al., 2019]).

Inference speedup. The reduction in model complexity in FLOPs that AAP achieves does translate to real speedup in model inference. We conducted inference experiments using PyTorch framework on Raspberry Pi 3B+, a widely used Internet-of-Things (IoT) platform. Table 6 shows our method significantly improves the inference speed.

Extension to multi-objective optimization. Our method can be readily extended to support *multiple objectives* and *multiple constraints* for optimizing the pruned model in terms of accuracy, size, and/or speed simultaneously. Table 7 shows two examples from pruning VGG-19 with CIFAR-10. In the first example, the objective is to minimize the accuracy loss under the constraints of 80% reduction in *both* FLOPs and parameters. In the second example, the objective is to maximize *both* the FLOPs reduction and parameters reduction given no more than 1% accuracy loss.

Table 6: Inference speedup on Raspberry Pi. For each type of model, the two rows are two pruned models from the same uncompressed model with different levels of parameters reduction.

Model and Dataset	Target	Acc. ↓ (%)	FLOPs. ↓ (%)	Speedup
ResNet-56 (CIFAR-10)	0%	-0.10	33.77	1.20×
		-0.09	56.33	1.49×
VGG-16 (CIFAR-10)	1%	-0.17	59.51	2.13×
		0.59	73.89	3.60×
ResNet-50 (Tiny-ImageNet)	5%	-3.92	35.06	1.01×
		-1.62	58.16	1.49×

Table 7: Multi-objective optimization on VGG-19 (CIFAR-10).

Optimization Objectives	Constraints	Acc. ↓ (%)	Params. ↓ (%)	FLOPs ↓ (%)
Minimize Accuracy Loss	Params. ↓ >80% and FLOPs ↓ >80%	1.20	83.11	80.60
Maximize FLOP Reduction and Params. Reduction	Acc. ↓ <1%	0.15	89.11	62.51

5 Conclusions

This paper proposes Automatic Attention Pruning (AAP), an adaptive, attention-based, structured pruning solution to automatically and efficiently generate small, accurate, and hardware-efficient models that meet diverse user requirements. We show that activation-based attention is a more precise indicator for identifying unimportant filters to prune than the commonly used weight magnitude value. We also offer an effective way to perform structured pruning in an adaptive process and find small and accurate subnetworks that are at the same time hardware efficient. Finally, we argue that automatic pruning is essential for pruning to be useful in practice, and propose an adaptive method that can automatically meet diverse user objectives in terms of model accuracy, size, and inference speed but without user intervention. Our results confirm that our solution outperforms existing structured pruning approaches by a large margin.

6 Acknowledgments

We thank the anonymous reviewers for their feedback. This work is partly supported by National Science Foundation awards CNS-1955593 and OAC-2126291 and an Amazon Machine Learning Research Award.

References

[Ashok et al., 2017] Ashok, A., Rhinehart, N., Beainy, F., and Kitani, K. M. (2017). N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

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

[Chen et al., 2022] Chen, T., Chen, X., Ma, X., Wang, Y., and Wang, Z. (2022). Coarsening the granularity: Towards structurally sparse lottery tickets. In *International Conference on Machine Learning*, pages 3025–3039. PMLR.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Dubey et al., 2018] Dubey, A., Chatterjee, M., and Ahuja, N. (2018). Coreset-based neural network compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 454–470.

[Frankle and Carbin, 2018] Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

[Frankle et al., 2019] Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2019). Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*.

[Gao et al., 2020] Gao, S., Huang, F., Pei, J., and Huang, H. (2020). Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1899–1908.

[Guo et al., 2020] Guo, J., Ouyang, W., and Xu, D. (2020). Multi-dimensional pruning: A unified framework for model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1508–1517.

[Guo et al., 2021] Guo, Y., Yuan, H., Tan, J., Wang, Z., Yang, S., and Liu, J. (2021). Gdp: Stabilized neural network pruning via gates with differentiable polarization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5239–5250.

[Han et al., 2015a] Han, S., Mao, H., and Dally, W. J. (2015a). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

[Han et al., 2015b] Han, S., Pool, J., Tran, J., and Dally, W. (2015b). Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143.

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

[He et al., 2018b] He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. (2018b). Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800.

[He et al., 2019] He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4340–4349.

[He et al., 2017] He, Y., Zhang, X., and Sun, J. (2017). Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397.

[Hill et al., 2017] Hill, P., Jain, A., Hill, M., Zamirai, B., Hsu, C., Laurenzano, M. A., Mahlke, S., Tang, L., and Mars, J. (2017). 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)*, pages 786–799.

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

[Kang and Han, 2020] Kang, M. and Han, B. (2020). Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*, pages 5122–5131. PMLR.

[Krizhevsky et al., 2009] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

[Le and Yang, 2015] Le, Y. and Yang, X. (2015). Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3.

[LeCun et al., 1990] LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605.

[Li et al., 2020a] Li, B., Wu, B., Su, J., and Wang, G. (2020a). Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *European Conference on Computer Vision*, pages 639–654. Springer.

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

[Liebenwein et al., 2019] Liebenwein, L., Baykal, C., Lang, H., Feldman, D., and Rus, D. (2019). Provable filter pruning for efficient neural networks. *arXiv preprint arXiv:1911.07412*.

[Lin et al., 2020] Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., and Shao, L. (2020). Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1529–1538.

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

[Liu et al., 2017] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744.

[Ma et al., 2021] Ma, X., Lin, S., Ye, S., He, Z., Zhang, L., Yuan, G., Tan, S. H., Li, Z., Fan, D., Qian, X., et al. (2021). Non-structured dnn weight pruning—is it beneficial in any platform? *IEEE transactions on neural networks and learning systems*, 33(9):4930–4944.

[Meng et al., 2020] Meng, F., Cheng, H., Li, K., Luo, H., Guo, X., Lu, G., and Sun, X. (2020). Pruning filter in filter. *arXiv preprint arXiv:2009.14410*.

[Molchanov et al., 2017] Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR.

[Oquab et al., 2015] Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2015). Is object localization for free?-weakly-supervised learning with convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 685–694.

[Paupamah et al., 2020] Paupamah, K., James, S., and Klein, R. (2020). Quantisation and pruning for neural network compression and regularisation. In *2020 International SAUPEC/RobMech/PRASA Conference*, pages 1–6. IEEE.

[Renda et al., 2020] Renda, A., Frankle, J., and Carbin, M. (2020). Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*.

[Simonyan et al., 2013] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

[Springenberg et al., 2014] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

[Tanaka et al., 2020] Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems*, 33:6377–6389.

[Tang et al., 2020] Tang, Y., Wang, Y., Xu, Y., Tao, D., Xu, C., Xu, C., and Xu, C. (2020). Scop: Scientific control for reliable neural network pruning. *arXiv preprint arXiv:2010.10732*.

[Wang et al., 2019] Wang, C., Grosse, R., Fidler, S., and Zhang, G. (2019). Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International Conference on Machine Learning*, pages 6566–6575. PMLR.

[Wang et al., 2021] Wang, W., Chen, M., Zhao, S., Chen, L., Hu, J., Liu, H., Cai, D., He, X., and Liu, W. (2021). Accelerate cnns from three dimensions: a comprehensive pruning framework. In *International Conference on Machine Learning*, pages 10717–10726. PMLR.

[Wang et al., 2020] Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B., and Hu, X. (2020). Pruning from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12273–12280.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

[Yang et al., 2019] Yang, H., Wen, W., and Li, H. D. (2019). Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*.

[You et al., 2019] You, Z., Yan, K., Ye, J., Ma, M., and Wang, P. (2019). Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1909.08174*.

[Zagoruyko and Komodakis, 2016] Zagoruyko, S. and Komodakis, N. (2016). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*.

[Zhao et al., 2019] Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W., and Tian, Q. (2019). Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2780–2789.

[Zhou et al., 2016] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929.

[Zhuang et al., 2020] Zhuang, T., Zhang, Z., Huang, Y., Zeng, X., Shuang, K., and Li, X. (2020). Neuron-level structured pruning using polarization regularizer. *Advances in Neural Information Processing Systems*, 33.

[Zhuang et al., 2018] Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., and Zhu, J. (2018). Discrimination-aware channel pruning for deep neural networks. *arXiv preprint arXiv:1810.11809*.

[Zoph et al., 2018] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.

A Appendix

In the supplementary materials, we first discuss the adaptive pruning policy with memory and FLOPs targets in Section A.1. Then we introduce the results of the additional experiments in Section A.2.

A.1 Adaptive Pruning Policy

A.1.1 Memory-constrained Adaptive Pruning

The Memory-constrained Adaptive Pruning Algorithm is shown in Algorithm 3.

Algorithm 3 Memory-constrained Adaptive Pruning

Input: Target Parameters Reduction $ParamTarget$
Output: A small pruned model with an acceptable model size
Initialize: $T = 0.0, \lambda = 0.01$.
for pruning round r ($r \geqslant 1$) **do**
  Prune the model using $T[r]$
  Rewind weights and the learning rate
  Train the pruned model, and calculate its remaining number of parameters $Param[r]$
  Calculate the parameter reduction: $ParamRed[r]$: $ParamRed[r] = Param[0] - Param[r]$
  if $ParamRed[r] < ParamTarget$ **then**
    if the changes of model size are within 0.1% for several rounds **then**
      Terminate
    else
      $\lambda[r+1] = \lambda[r]$
      $T[r+1] = T[r] + \lambda[r+1]$
    end if
  else
    Find the last acceptable round k
    if k has been used to roll back for several times **then**
      Mark k as unacceptable
      Go to Step 17
    else
      Roll back model weights to round k
      $\lambda[r+1] = \lambda[r]/2.0^{(C+1)}$ (C is the number of times for rolling back to round k)
      $T[r+1] = T[k] + \lambda[r+1]$
    end if
  end if
end for

A.1.2 FLOPs-constrained Adaptive Pruning

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

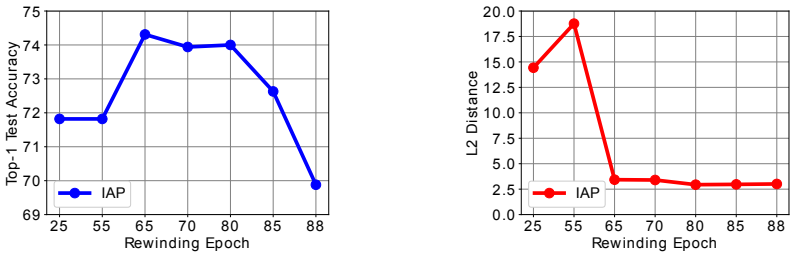
Algorithm 4 FLOPs-constrained Adaptive Pruning

Input: Target FLOPs Reduction $FLOPsTarget$
Output: A small pruned model with an acceptable FLOPs
Initialize: $T = 0.0, \lambda = 0.01$.
for pruning round r ($r \geqslant 1$) **do**
  Prune the model using $T[r]$
  Rewind weights and the learning rate
  Train the pruned model, and calculate its remaining FLOPs $FLOPs[r]$
  Calculate the parameters reduction: $FLOPsRed[r]$: $FLOPsRed[r] = FLOPs[0] - FLOPs[r]$
  if $FLOPsRed[r] < FLOPsTarget$ **then**
    if the changes of FLOPs are within 0.1% for several rounds **then**
      Terminate
    else
      $\lambda[r+1] = \lambda[r]$
      $T[r+1] = T[r] + \lambda[r+1]$
    end if
  else
    Find the last acceptable round k
    if k has been used to roll back for several times **then**
      Mark k as unacceptable
      Go to Step 17
    else
      Roll back model weights to round k
      $\lambda[r+1] = \lambda[r]/2.0^{(C+1)}$ (C is the number of times for rolling back to round k)
      $T[r+1] = T[k] + \lambda[r+1]$
    end if
  end if
end for

A.2 Additional Experiments

A.2.1 The Effect of Rewinding Epoch

To understand how the rewinding impacts the accuracy of the pruned models, we analyze *stability to pruning*, 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 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 4a shows the Top-1 test accuracy of the pruned ResNet-50 with a parameter reduction of 16.26% on ImageNet when the learning rate is rewound to different epochs, and Figure 4b 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 the L2 distance closely follows this pattern, showing a large distance for early training epochs and a small distance for later training epochs. Our findings show that rewinding to 75%-90% of training time leads to good accuracy.

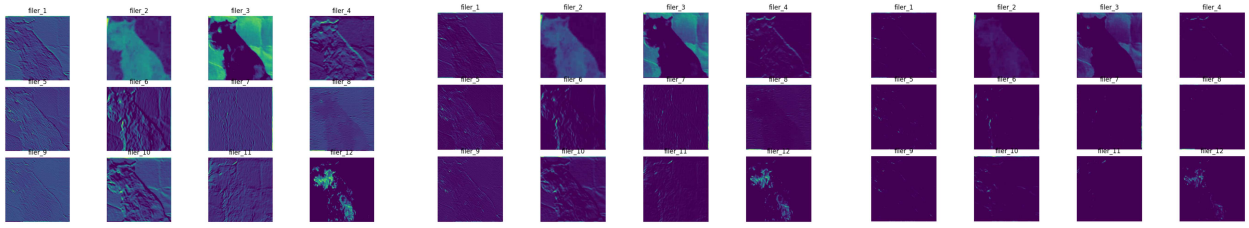


(a) Top-1 Test Accuracy (b) Stability to Pruning

Figure 4: The effect of the rewinding epoch (x-axis) on (a) Top-1 test accuracy, and (b) pruning stability, for pruned ResNet-50 with a parameter reduction of 16.26% on ImageNet.

A.2.2 The Effect of Attentions

Figure 5 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 it promotes the effectiveness of the pruning by enabling the gap between the mean values of the useful and useless filters to be large.

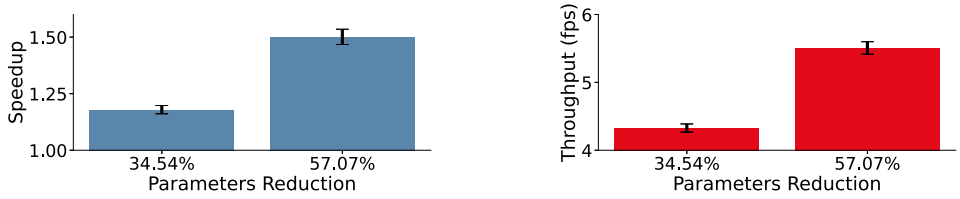


(a) p=1 (b) p=2 (c) p=4

Figure 5: Attentions of each filter of the first convolution layer of ResNet-50 on ImageNet with different values of p ($p = 1, 2, 4$).

A.2.3 Inference Speedup on CPU

Figure 6 shows the speedup of ResNet-50 with a parameter reduction of 34.54% and 57.07%, respectively, on one Intel(R) Xeon(R) Silver 4215R CPU. We run it for 10 trails. The input image size is 224×224. The average throughput of the original ResNet-50 for 10 trails is 3.67fps.

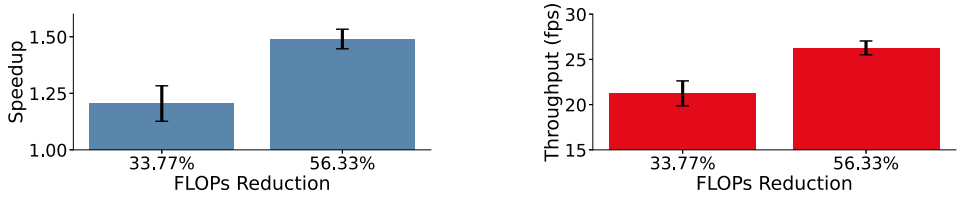


(a) Speedup (b) Throughput

Figure 6: The illustration of (a) the speedup and (b) the throughput of ResNet-50 with a parameter reduction of 34.54% and 57.07%, respectively, on one Intel(R) Xeon(R) Silver 4215R CPU. We run it for 10 trails. The input image size is 224×224. The average throughput of the original ResNet-50 is 3.67fps.

A.2.4 Inference Speedup on Raspberry Pi

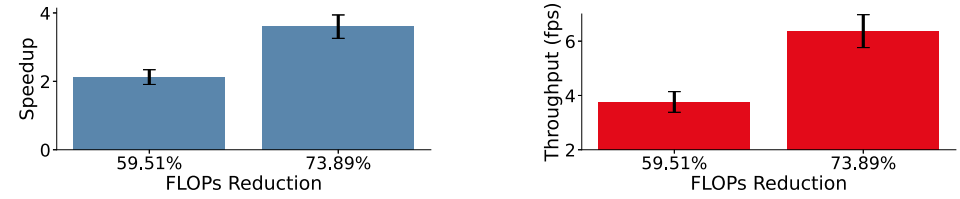
ResNet-56 on CIFAR-10. Figure 7 shows the speedup of ResNet-56 with a FLOP reduction of 33.77% and 56.33%, respectively, on Raspberry Pi 3B+. We run it for 10 trails. The input image size is 32×32. The average throughput of the original ResNet-56 for 10 trails is 17.63fps.



(a) Speedup (b) Throughput

Figure 7: The illustration of (a) the speedup and (b) the throughput of ResNet-56 with a FLOP reduction of 33.77% and 56.33%, respectively, on Raspberry Pi 3B+. We run it for 10 trails. The input image size is 32×32. The average throughput of the original ResNet-56 is 17.63fps.

VGG-16 on CIFAR-10. Figure 8 shows the speedup of VGG-16 with a FLOPs reduction of 59.51% and 73.89%, respectively, on Raspberry Pi 3B+. We run it for 10 trails. The input image size is 32×32. The average throughput of the original VGG-16 for 10 trails is 1.77fps.



(a) Speedup (b) Throughput

Figure 8: The illustration of (a) the speedup and (b) the throughput of VGG-16 with a FLOP reduction of 59.51% and 73.89%, respectively, on Raspberry Pi 3B+. We run it for 10 trails. The input image size is 32×32. The average throughput of the original VGG-16 is 1.77fps.

A.2.5 Attention Distributions

Figure 9 shows the distribution of the attention values of each convolutional layer of the original ResNet-50 and pruned ResNet-50 with a parameter reduction of 96.31% on ImageNet. Specifically, given one batch of images, the models do the inference once. We first measure the attention value ($p = 1$) of each filter for each image, and then we calculate its average attention values for one batch of images.

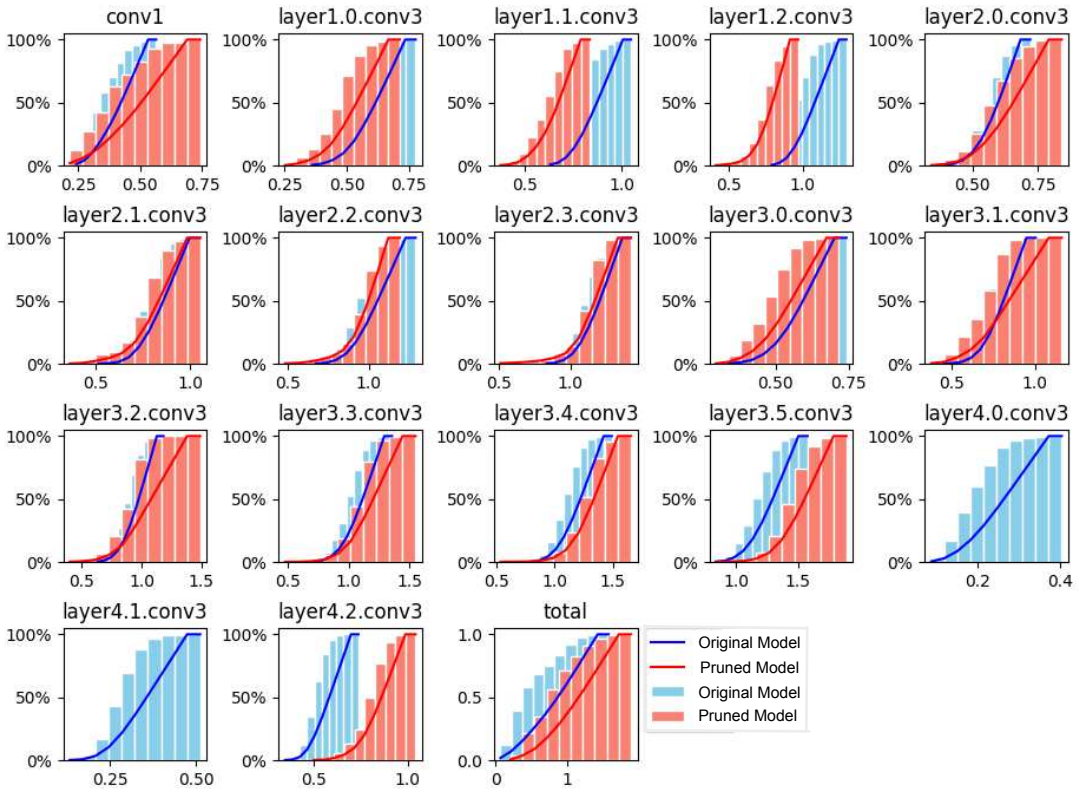


Figure 9: The the distribution of the attention values of each convolutional layer of the original ResNet-50 and the pruned ResNet-50 with a parameter reduction of 96.31% on ImageNet.