Design and analysis of hardware friendly pruning algorithms to accelerate deep neural networks at the edge

Christin David Bose[*]
chris241@purdue.edu
Purdue University
Indiana, USA

Mostafa Elshoushi
melhoushi@meta.com
Meta AI
Toronto, Canada

Syed Shakib Sarwar
shakib7@purdue.edu
Meta Reality Lab Research
Redmond, WA, USA

Yuecheng Li
yuecheng.li@meta.com
Meta Reality Lab Research
Pittsburg, PA, USA

Reid Federick Pinkham
pinkhamr@meta.com
Meta Reality Lab Research
Redmond, WA, USA

Mia Kasperek
miaskomski22@meta.com
Meta Reality Lab Research
Redmond, WA, USA

Ziyun Li
liziyun@meta.com
Meta Reality Lab Research
Redmond, WA, USA

Barbara Da Salvo
barbarads@meta.com
Meta Reality Lab Research
Burlingame, CA, USA

ABSTRACT

Employing unstructured pruning is very popular to get state-of-the-art model compression of convolutional neural network weights. While unstructured pruning results in good model accuracy, it is challenging for hardware architectures to exploit unstructured sparsity for speedups and power savings during model inference. Structured pruning, on the other hand, readily translates to model inference speedups and power reduction due to its hardware-friendly nature but typically results in inferior model accuracy when compared with unstructured pruning. Model pruning is performed by removing network weights according to some criteria. As there are several structured and unstructured pruning criteria being proposed in literature, it is often unclear which criteria offers the best performance in hardware. In this work, we generate the accuracy-sparsity-latency pareto curve for several state-of-the-art filter pruning criteria and generate insights based on an edge-based DNN accelerator. We also propose combining various granularities of pruning and evaluate the benefits. These insights will be useful to prune deep learning workloads for inference at the edge subject to a given accuracy and compute budget.

KEYWORDS

Pruning, deep neural networks, sparsity, edge hardware accelerator

ACM Reference Format:
Christin David Bose, Mostafa Elshoushi, Syed Shakib Sarwar, Yuecheng Li, Reid Federick Pinkham, Mia Kasperek, Ziyun Li, and Barbara Da Salvo. 2022. Design and analysis of hardware friendly pruning algorithms to accelerate deep neural networks at the edge. In *Proceedings of tinyML Research Symposium (tinyML Research Symposium'23).* ACM, New York, NY, USA, 7 pages.

[*]Work done during employment at Meta

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
tinyML Research Symposium'23, March 2023, San Jose, CA
© 2022 Copyright held by the owner/author(s).

1 INTRODUCTION

Deep neural networks have seen an explosion of use cases over the past decade and typically represent the state of the art solutions for artificial intelligence at the edge in many fields. Convolutional layers (CNNs) constitute a significant portion of these deep neural networks. However, CNNs have large compute and memory footprint that restricts deployment in scenarios which resources are limited. For example, many edge deployment scenarios need to respect stringent form factor, computing, power and thermal requirements. Novel distributed on-sensor compute architecture, coupled with new semiconductor technologies and, most importantly, a full hardware-software co-optimization are the solutions to achieve these strict requirements.

There are a plethora of solutions in literature that enable aggressive hardware-algorithm co-optimization. For example, non-traditional quantization and sparse model generation schemes have been extensively studied to generate models with low memory footprint and good accuracy. However, most hardware acceleration platforms do not have support to adapt to these non-traditional techniques. While some hardware platforms like NVIDIA A100 GPUs have sparse tensor cores to enable structured sparsity, the onus is on the programmer to use necessary libraries and generate model weights in a given format (for example 2:4 sparsity) in order to utilize the hardware sparsity support. Even after employing unstructured sparsity, the savings are sub-linear as the hardware overhead associated with such techniques is non-negligible. For example, NVIDIA's 2:4 sparsity results in an end-to-end network speedup of at most 1.5x [20] for common models despite having 50% sparsity. Hence, this work focuses on developing more hardware friendly techniques that can be readily applied with minimal assistance from the programmer or compiler. In short, this work targets structured or semi-structured forms of pruning to accelerate machine learning models at the edge.

In order to generate machine learning models with structured sparsity, there exists several criteria in the literature to rank filters or kernels ([12],[8],[17],[18]). We note that while it is easy to specify a filter or kernel ranking criteria, it is not clear which ranking algorithm will be best (in terms of accuracy and/or hardware friendliness) to generate such sparse models.

This paper makes the following contributions.

- We compare the hardware friendliness of several state of the art filter pruning criteria subject to a given accuracy degradation budget.
- We introduce a structured sparsity pruning algorithm that learns a differential mask for pruning with a novel loss formulation that aims to minimize the difference between the output activations of the pruned and dense model in addition to the task loss.
- We further demonstrate combining structured and semi structured pruning in order to get additional benefits in hardware acceleration.
- Finally, we deploy the pruned models on an edge hardware accelerator (ARM Ethos U55) and compare the inference latency of various pruning criteria. As different pruning techniques generate differently sized models (even for the same model compression ratio), we generate accuracy-latency curves for our proposed method as well as the other methods from literature. Such a comparison yields insights into the hardware friendliness of different pruning techniques for a given compression ratio or latency budget.

2 PRELIMINARIES

In this section, we describe some common terminology used in literature for structured pruning of CNNs.

Let I_l and W_l be the input features (or input activations) and weights of a convolution layer l, respectively, where $I_l \in \mathbb{R}^{c_{l-1}\times w_l \times h_l}$, $W_l \in \mathbb{R}^{c_l \times c_{l-1} \times k_l \times k_l}$, and c_l is the number of filters in layer l. A typical CNN block consists of a convolution operation ($*$), batch normalization (BN), and an activation function (g) such as the commonly used ReLU. Without loss of generality, we ignore the bias term due to the bias term included in the BN, thus, the output feature map O_l (output activations) can be written as $O_l = g(BN(I_l * W_l))$.

We note that the above example convolves the input activations with c_l filters each of size $c_{l-1} \times k_l \times k_l$ in order to get c_l output channels. By pruning away entire filters of size $c_{l-1} \times k_l \times k_l$, we realize that the corresponding output channel is also removed. We define each filter of a CNN layer as consisting of different kernels. In this example, we have c_l filters each having c_{l-1} kernels of size $k_l \times k_l$. In subsequent sections, we primarily refer to structured pruning as removing different filters (filter pruning) and semi-structured pruning as removing different kernels of a filter (kernel pruning).

As pruning of each filter removes an output channel, the corresponding input channels of each filter of the next convolutional layer can also be removed. This is depicted in the figure below:

We note this as filter pruning induced kernel pruning and this will be one of our optimizations in generating smaller and more compact models for hardware deployment.

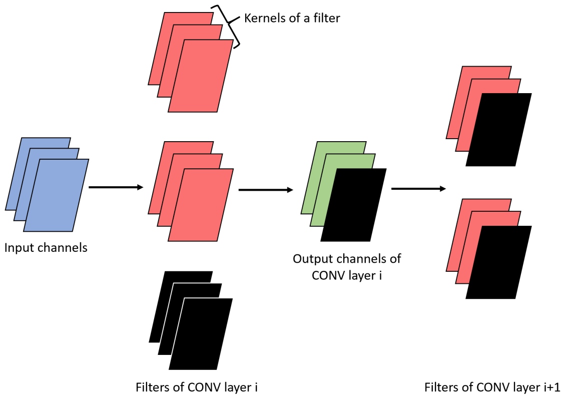


Figure 1: Filter pruning induced kernel pruning. Note: Greyed out channels denote pruned filters/kernels

3 RELATED WORK

3.1 Pruning

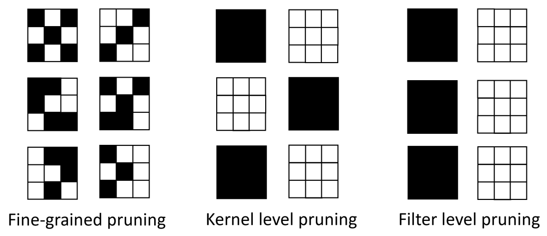


Figure 2: Overview of pruning granularity

Static pruning is the process of removing weights during the offline training stage and applying the same model to all input samples. Unstructured weight pruning ([12],[8],[17],[18]) is the most fine-grained as it removes individual weights. While unstructured pruning could help in model compression, extensive hardware/compiler support is needed to get speedup from the compression. Structured or semi-structured pruning is becoming a popular alternative where whole filters or blocks of weights are being removed based on a ranking criteria. Figure 2 depicts selected granularities of pruning commonly deployed. Dynamic pruning on the other hand proposes to skip irrelevant computations while performing inference. Thus, dynamic pruning processes different routes per input sample ([11], [15], [5], [21])

3.2 Neural Architecture Search (NAS)

Recently, NAS has increasingly been used to find more efficient sparse models. While filter pruning just involves the removal of filters (or subsequent channels), NAS can optimize for a broader set of parameters such as number of layers, kernel size, bitwidth, number of neurons per layer, etc. Typically, when used for the same filter pruning purpose, NAS is more computationally expensive when compared to other filter pruning techniques. Sparse model generation through pruning is particularly useful in low resource settings where GPU hours are limited due to limited hardware or time sharing requirements. For example, findings from [7] claim that on average the GPU-hours incurred during pruning is about 4x less than that consumed by a full-scale NAS.

3.3 Quantization

Several works ([3], [6], [13], [14]) have employed various forms of weight and activation quantization in order to reduce memory footprint and improve speedup. We leave the joint optimization of sparsity and quantization as future work.

4 HOW TO PRUNE A FILTER?

4.1 Pruning methods from Literature

In order to remove groups of weights such as a filter, many methods have been proposed based on a filter selection criteria ([9], [10], [16], [18], [19], [23]). Molchanov et al. [19] learns a global importance measure of a filter and calculates the global importance score based on a Taylor approximation on the network's weights where a first order approximation of a filter's gradients and its norm are used to calculate a filter's importance metric. Other works such as [16] and [22] propose to introduce a sparsity loss in addition to the task loss as a regularization term and then prune filters whose criterion are less than a threshold. Yu et al [23] uses the Hessian based trace to find insensitive filters in a model. From a comparison across several performance numbers reported in the literature, we mainly compare our methods against the Taylor [19] and Hessian based approach (HAP) for structured pruning [23].

4.2 Self learning pruning critera

In this section, we describe the mechanism for learning the importance of each filter (or kernel). The following description applies to pruning of filters but the same methodology can be applied to pruning at other granularities (for e.g.: kernel). We then describe the design of filter selection based on the importance and the loss formulation that enables the learning of the importance value.

Our proposed method learns a score value for each filter which is then translated into a binary mask that depicts whether a filter is chosen for pruning or not.

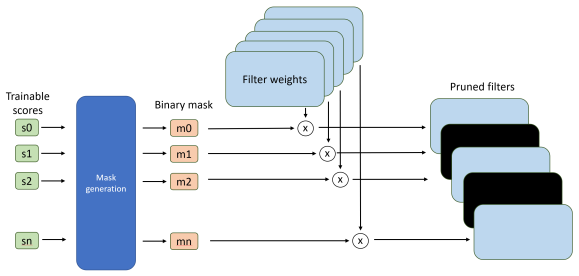


Figure 3: Supermask based filter pruning. Greyed-out filters indicate pruned filters

Taking motivation from the selection approach used in [4], we employ a cumulative sorting based selection of the filters based on the trained score values. The k filters that are to be kept are calculated based on a global hyperparameter r. For each CNN layer, k is the number of filters kept such that the cumulative mass of the sorted score values reaches r. The same hyperparameter r is used for all layers. However, each layer will have a different pruning ratio depending on the score values. The advantage of this approach is to avoid manually setting the pruning ratio of each layer without resorting to determining the layer sensitivity or a global filter importance measure. Sensitivity analysis is known to be computationally expensive as the layers grow deeper.

```
**Input:** $s_1, s_2, ...s_n, r$
**Output:** $m_1, m_2, \ldots, m_n$
$m[1:n] \leftarrow ones(1,n)$;
$scores \leftarrow sigmoid(scores)$;
$normalize \leftarrow scores/SUM(scores)$;
$sorted, idx \leftarrow SORT(normalize, descending = True)$;
$cumulative \leftarrow CUMSUM(sorted)$;
$pruneidx \leftarrow WHERE(cumulative > r)$;
$m[pruneidx] \leftarrow 0$;
```
Algorithm 1: Binary mask generation

Loss function

In order to learn which filters are important, we augment the task loss with an auxiliary loss that serves as a proxy for the usefulness of a given filter. The overall training objective is as follows:

$$\min_W L_{total} = L_{ent}(f_n(x;W), y_k) + L_{aux} \tag{1}$$

where $L_{ent}(f_n(x;W), y_k)$ denotes the cross-entropy task loss and L_{aux} is the auxiliary loss.

The auxiliary loss is calculated as:

$$L_{aux} = RMSE(g(BN(I_l * W_l)),\ g(BN(I_l * mask(W_l)))) \tag{2}$$

where $RMSE$ is the root mean square error, $g(BN(I_l * W_l))$ denotes the output activations of a dense model and $g(BN(I_l * mask(W_l)))$ denotes the output activations after pruning the filters.

We train solely on the auxiliary loss during the initial 50 epochs in order to distill the important filters from a pretrained model. We then subsequently train the model only on the task loss (100 epochs).

5 EXPERIMENTS AND ANALYSIS

Pruning

We base our experiments on the ResNet20 models on CIFAR100 and the ResNet50 and MobileNetV2 models on ImageNet dataset. We compare our pruning methods to other popular pruning methods in literature such as Taylor gate based pruning [19] and Hessian aware pruning [23]. The points of comparison are on two fronts: accuracy and inference latency. For all points of comparison, we have used the same experimental settings (*e.g.*, hyperparameters) used in [19], [23] in order to get the best possible accuracy with the respective pruning criteria. Regarding skip connections such as commonly found in ResNet models, we pruned the both paths of the skip connections by the same fraction so that tensor additivity is satisfied (Figure 4).

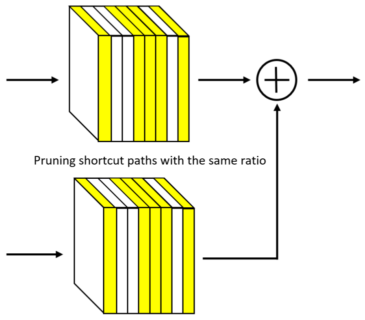


Figure 4: Pruning of residual connections

Hardware deployment

The target hardware chosen is the ARM based Ethos U55 [2] which is an inference accelerator. The latency estimates for the models are obtained after compiling the models with the ethos-u-vela compiler [1] provided by ARM in the open source community. We generated the TFLite models from PyTorch using PyTorch-ONNX-TFLite flow to generate required quantized network format for deploying in the ARM NPU. As our target use case is for inference at the edge (such as for AR/VR or IoT like workloads), we have used a batch size of 1 for all our latency estimates. During the pruning phase of the algorithm, the pruned filters are merely zeroed out while the model architecture is unchanged. Naively, compiling this "pruned" model using the ethos-u-vela compiler results in memory footprint and read cycle savings as some of the zeroed out weights can now be compressed. However, the NPU cycles remain unchanged (as the weights are decompressed before the computation) and the overall latency savings is limited. In order to evaluate the latency of a truly pruned model, we indeed create a smaller model by totally removing pruned filters and kernels (as illustrated in section 2). Some of the key specifications of the hardware accelerator assumed is shown in Table 1.

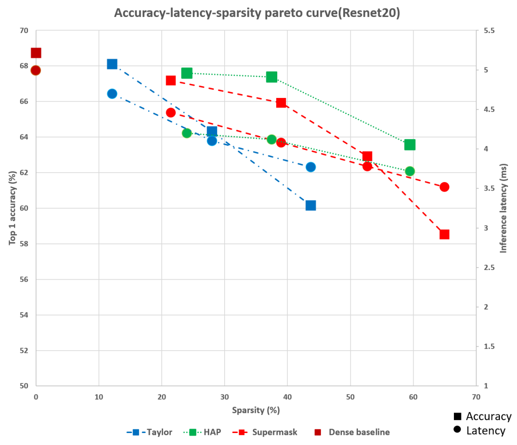
Metric	Value
System clock	500Mhz
Number of Processing Elements (PEs)	32-256
Design peak SRAM bandwidth	4.00GB/s
Design peak On-chip Flash	4.00GB/s

Table 1: Hardware accelerator specifications

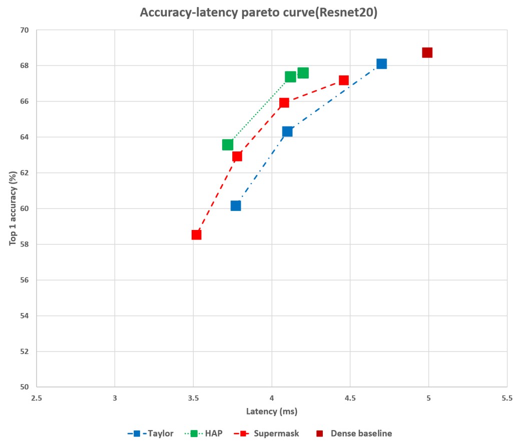
It should be noted that the SRAM is used for runtime data storage (like activations) while the on-chip flash memory is used for weight storage. The PyTorch models are compiled with performance optimization as an objective (least inference time) i.e. the compilation process prioritises a higher runtime performance over a lower peak SRAM usage. In order to maintain an acceptable frame rate for edge inference, we have used 32 PEs for ResNet20 and 256 PEs for ResNet50 and MobileNetV2.

5.1 Experiments on CIFAR100

5.1.1 ***ResNet20.*** We plot the accuracy-sparsity-latency curve for ResNet20. At low levels of sparsity (<20%), we see that Taylor based pruning offers minimal accuracy degradation. However, at moderate to high levels of sparsity (20%-50%), Taylor based pruning has strong roll off in accuracy as compared to Hessian based pruning. Hessian based pruning also offers better accuracy at similar latency when compared to the self-learning filter pruning method (Supermask)



(a)



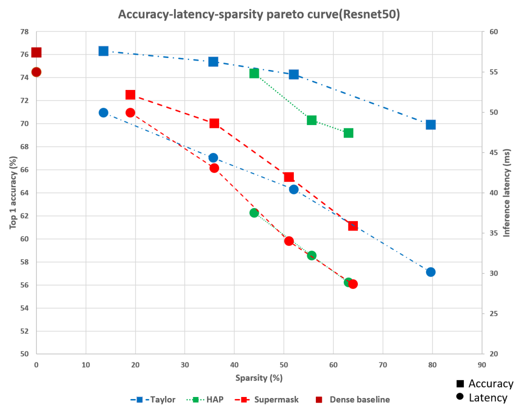
(b)

Figure 5: ResNet20 on CIFAR100

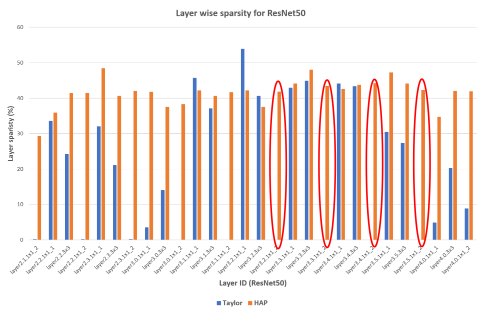
We notice that the same sparsity level results in a pruned model with varied inference latencies depending on the pruning criteria. This is because different pruning criteria may choose to remove differently sized filters by different ratios. We also plot the accuracy-latency tradeoff curve in Figure 5(b). At strict latency constraints, Hessian based pruning is able to give the best model accuracy for a given latency.

5.2 Experiments on Imagenet

5.2.1 ***ResNet50.*** We plot the accuracy-sparsity-latency curve for the ResNet50 model on Imagenet for the pruning strategies as discussed earlier (Figure 6a). We first notice that for a given sparsity level, there is a significant variance in the latency of the pruned models from different pruning criteria. Taylor pruning offers minimal accuracy degradation at the expense of high latency compared to other pruning methods in consideration. On comparing the pruned models from Taylor and the other approaches, it was found that Taylor pruning leaves a lot of 1x1 kernels intact. 1x1 kernels are well known to be inefficient on accelerators. This explains the poor latency behavior of Taylor based pruned models. Figure 6(b) compares the layerwise wise pruning strategy (for selected ResNet50 layers) of the Taylor and the HAP methods at an iso-sparsity level. While the Taylor based criteria hardly prunes the circled 1x1 kernels, HAP strongly prunes away these filters. The Supermask based method offers comparable latency as compared to the Hessian based approach (as it prunes away 1x1 kernels as well) but results in inferior model accuracy when compared to other approaches under consideration.

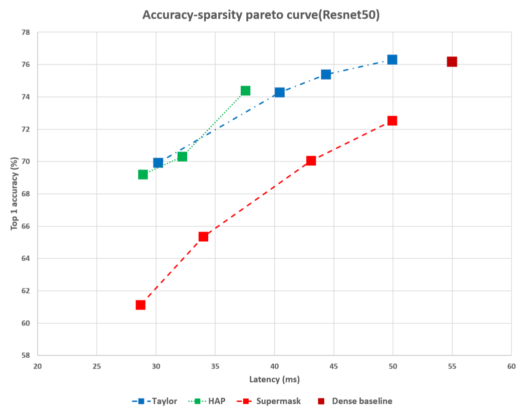


(a)



(b) Note: for the circled layers, the resultant sparsity of the Taylor scheme is nearly 0.

Figure 6(c) depicts the accuracy per latency budget for ResNet50 on ImageNet. At strict latency budgets, Taylor pruned models performs marginally better in terms of accuracy. It is to be noted that proportional increases in sparsity levels may result in disproportional increases in latency (depending on the model mapping strategy, layerwise pruning ratio etc)

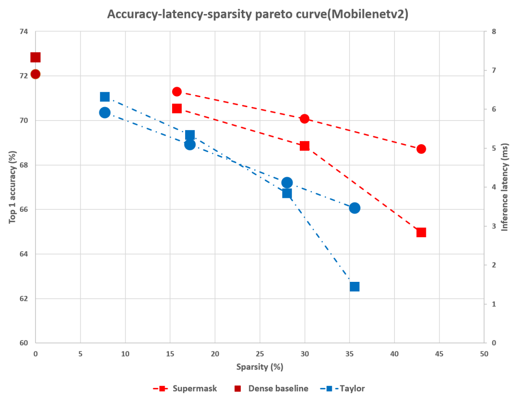


(c)

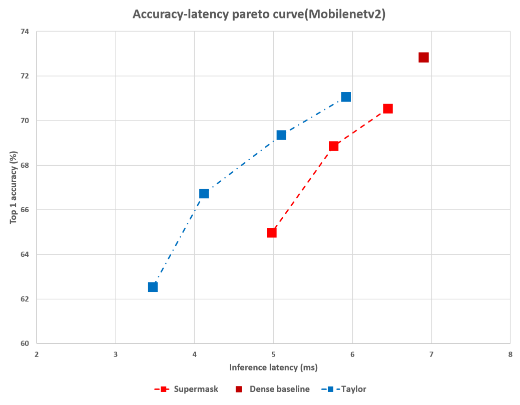
Figure 6: ResNet50 on ImageNet

5.2.2 *Mobilenetv2*.

In this section, we compare the performance of the pruning algorithms on compact machine learning models that are commonly used for deep learning on the edge. We plot the accuracy-sparsity-latency curve for MobileNetV2 in Figure 7a. As MobileNetV2 is already a compact model, we see steep accuracy degradation even at low levels of sparsity (eg: >10%). We notice pruned models using Taylor based criteria consistently underperform in accuracy when compared to the self-pruning criteria. As the Taylor criteria is based on a global filter ordering of importance [19], it was found that there is high variance in the inter-layer pruning ratio (ie. some layers are pruned much more strongly than others). This explains the drop in accuracy although at improved latency for the Taylor-based pruned model. We tried generating pruned models for MobileNetV2 based on [23] but their code base is old and was not amenable to readily pruning new model architectures. While Taylor based pruning results in a worse accuracy for a given sparsity level, the accuracy per latency budget is higher than the supermask method as shown in figre 7b.



(a)



(b)

Figure 7: MobileNetV2 on ImageNet

5.3 Choice of pruning criteria

In this subsection, we summarize our findings on sparse model generation using different pruning criteria. The first key observation is that no pruning criteria consistently outperforms the others for all models on a given hardware platform. While Taylor based filter pruning offers competitive accuracy at low levels of sparsity, the accuracy degradation at moderate to high levels of sparsity depends on the kind of model. Inference latency can also significantly vary between models generated from different pruning criteria at the same sparsity level. For instance, 1x1 kernels are pruned lightly by the Taylor criteria which results in limited speedup on edge hardware. On the other hand, compact models like MobileNetV2 results in some layers being more strongly pruned by the Taylor criteria as compared to others which degrades accuracy. Hessian based pruning offers good accuracy and latency at moderate levels of sparsity. However, it is to be noted that since the calculation of the Hessian is a second order method, the cost of sparse model generation is higher than the other studied methods. Constraints on model turn-around time are commonly found while applying sparsification to production models.

5.4 Combining structured and semi-structured filter pruning

In this section, we present a use case for combining various granularities of pruning. For the sake of illustration, we use the supermask method discussed in section 4.2. However, the principle can be extended to other pruning criteria as well. The motivation here is to remove both filters (structured sparsity) and kernels (semi-structured sparsity). While filter pruning can be readily translated to speedup in hardware by creating a smaller model, getting hardware speedup from kernel pruning requires changes to the model representation and/or the hardware compute engine which may not be a trivial change to the model user. In this example, we just zero out the kernel weights and expect to see benefits in model compression from the compiler. This optimization enables fewer memory read cycles as the weights are read in the compressed format and decompressed on-the-fly during the processing using dense compute engine, As the memory and compute cycles are pipelined in the accelerator, savings due to the reduction in the memory cycles may not be evident from the overall latency. However, since kernel pruning enables storing weights in a compressed format, memory leakage and read energy savings can be expected even with same overall latency. Since, the assumed vela compiler does not provide such energy stats, we ignore the estimation of potential energy savings from kernel pruning. While both filter pruning and kernel pruning offers advantages, the joint exploration of these two approaches for model optimization is left as future work.

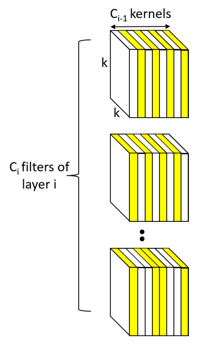


Figure 8: Kernel pruning illustrated in layer i. 50% of the kernels are pruned as an example.

Figure 8 illustrates an example of kernel level pruning (50% of kernels of each filter are pruned away). Table 2 depicts the model compression and accuracy by doing kernel level pruning in addition to 10% filter pruning (ie. in each layer 10% of filters are pruned) for the ResNet20 model on CIFAR100 dataset. We can see that weights can be compressed by 15-25% with negligible drop in accuracy. Further improvements in accuracy can be expected by pruning filters and kernels non-uniformly (eg: different prune ratios in each layer obtained through Algorithm 1).

Kernels pruned (%)	Accuracy (%)	Size (KiB)	Compression
0	67.180	252.64	0.893
10	67.250	235.73	0.834
20	66.100	214.56	0.759
30	66.475	192.20	0.680
40	65.475	172.67	0.611

Table 2: Mixed filter+kernel pruning. Baseline accuracy of ResNet20 dense model on CIFAR100 is 68.81%

6 CONCLUSION

Model pruning is a very popular optimization that aims at minimizing model footprint such as latency/energy/memory while trying to maintain accuracy. Structured pruning of deep neural networks can be readily translated to speedup on hardware acceleration platforms (such as GPUs, NPUs etc) while unstructured pruning requires hardware modification and/or compiler support to get benefits. Filter pruning of convolutional layers is an example of structured pruning that has received immense attention in the research community. While, there are several filter pruning criteria in the literature, it is often unclear which pruning criteria is best suited for a given hardware and accuracy budget. This works aims to study the hardware friendliness of several state of the art filter pruning criteria subject to a within sparsity level and accuracy degradation bucket. We find that the choice of the optimal pruning criteria varies depending on the model architecture, hardware platform and constraints such as accuracy/latency budget. We also show benefits of kernel pruning which can be applied in addition to filter pruning. These insights will be useful to model optimization engineers to select a suitable pruning criteria for a given model accuracy and latency budget.

REFERENCES

[1] ARM. [n. d.]. *ethos-u-vela.* https://review.mlplatform.org/plugins/gitiles/ml/ethos-u/ethos-u-vela.

[2] ARM. [n. d.]. *Ethos-U55.* https://developer.arm.com/Processors/Ethos-U55.

[3] Mostafa Elhoushi, Farhan Shafiq, Ye Henry Tian, Joey Yiwei Li, and Zihao Chen. 2019. *DeepShift: Towards Multiplication-Less Neural Networks.* https://arxiv.org/abs/1905.13298.

[4] Sara Elkerdawy, Mostafa Elhoushi, Hong Zhang, and Nilanjan Ray. 2022. Fire Together Wire Together: A Dynamic Pruning Approach with Self-Supervised Mask Prediction. *CVPR* (2022).

[5] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. 2018. *Dynamic channel pruning: Feature boosting and suppression.* https://arxiv.org/abs/1810.05331.

[6] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. *A Survey of Quantization Methods for Efficient Neural Network Inference.* https://arxiv.org/abs/2103.13630.

[7] Sayan Ghosh, Karthik Prasad, Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Graham Cormode, and Peter Vajda. [n. d.]. *Pruning Compact ConvNets For Efficient Inference.* https://openreview.net/forum?id=_gZ8dG4vOr9.

[8] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *In Advances in Neural Information Processing Systems (NIPS)* (2015).

[9] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, , and Yi Yang. 2018. *Soft filter pruning for accelerating deep convolutional neural networks.* https://arxiv.org/abs/1808.06866.

[10] Yihui He, Xiangyu Zhang, and Jian Sun. [n. d.]. Channel pruning for accelerating very deep neural networks. *In Proceedings of the IEEE International Conference on Computer Vision, pages 1389–1397* ([n. d.]).

[11] Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G Edward Suh. 2018. *Channel gating neural networks.* https://arxiv.org/abs/1805.12549.

[12] Michael Carbin Jonathan Frankle. 2018. *The lottery ticket hypothesis: Finding sparse, trainable neural networks.* https://arxiv.org/abs/1803.03635.

[13] Edward H. Lee, Daisuke Miyashita, Elaina Chai, Boris Murmann, , and S. Simon Wong. 2017. LogNet: Energy-efficient neural networks using logarithmic computation. *In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 5900–5904* (2017). https://doi.org/10.1109/ICASSP.2017.7953288.

[14] Yuhang Li, Xin Dong, and Wei Wang. 2019. *Additive Powers-of-Two Quantization.* https://arxiv.org/abs/1909.13144.

[15] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime neural pruning. *n Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017).

[16] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. *In Proceedings of the IEEE International Conference on Computer Vision, pages 2736–2744* (2017).

[17] Christos Louizos, Max Welling, and Diederik P Kingma. 2017. *Learning sparse neural networks through l0 regularization.* https://arxiv.org/abs/1712.01312.

[18] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. A filter level pruning method for deep neural network compression. *In Proceedings of the IEEE international conference on computer vision* (2017).

[19] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 11264– 11272* (2019).

[20] Jeff Pool. [n. d.]. *Accelerating sparsity in the NVIDIA Ampere Architecture.* https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf.

[21] Yulong Wang, Xiaolu Zhang, Xiaolin Hu, Bo Zhang, and Hang Su. 2020. Dynamic network pruning with interpretable layerwise channel selection. *In Proceedings of the AAAI Conference on Artificial Intelligence* (2020).

[22] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. *n Advances in neural information processing systems, pages 2074–2082* (2016).

[23] Shixing Yu, Zhewei Yao, Amir Gholami, Zhen Dong, Sehoon Kim, Michael W. Mahoney, and Kurt Keutzer. 2022. Hessian-Aware Pruning and Optimal Neural Implant. *WACV* (2022).