

Leveraging Structured Pruning of Convolutional Neural Networks

Hugo Tessier^{*†}, Vincent Gripon[†], Mathieu Léonardon[†], Matthieu Arzel[†], David Bertrand^{*}, Thomas Hannagan^{*}

^{*}*Stellantis*, Vélizy-Villacoublay, France

{1, 5, 6}@stellantis.com

[†]*IMT Atlantique*, Lab-STICC, UMR CNRS 6285, F-29238 Brest, France

{1, 2, 3, 4}@imt-atlantique.fr

Abstract—Structured pruning is a popular method to reduce the cost of convolutional neural networks, that are the state of the art in many computer vision tasks. However, depending on the architecture, pruning introduces dimensional discrepancies which prevent the actual reduction of pruned networks. To tackle this problem, we propose a method that is able to take any structured pruning mask and generate a network that does not encounter any of these problems and can be leveraged efficiently. We provide an accurate description of our solution and show results of gains, in energy consumption and inference time on embedded hardware, of pruned convolutional neural networks.

Index Terms—Deep Learning, Compression, Pruning, Energy, Inference, GPU

I. INTRODUCTION

Deep neural networks are at the state of the art in many domains, such as computer vision. For instance, convolutional neural networks are used to tackle different tasks such as classification [17] or semantic segmentation [16]. However, their cost in energy, memory and latency is prohibitive on embedded hardware, and this is why many works focus on reducing their cost to fit targets with limited resources [1].

The field of deep neural networks compression counts multiple types of method, such as quantization [3] or distillation [9]. The one we focus on in this article is pruning [5], that involves removing unnecessary weights from a network. Pruning is a popular technique that presents many challenges, including that of finding the most adequate type of sparsity to be leveraged on hardware [13].

To focus on the theoretical approach of studying the impact of removing weights from the network’s function on its accuracy, many papers only remove weights by putting their value to zero. However, this does not reduce the cost of networks and only provides a rough estimate of network compression in terms of memory. Leveraging pruning to get gains on hardware is actually not a trivial task. Pruning isolated weights [5] (“non-structured pruning”) produces sparse matrices, that are difficult to accelerate [13]. Pruning entire convolution filters (a.k.a. “structured pruning”) is more easily exploitable, but the input and output dimensions of layers are altered, which can induce many problems in networks, especially those including long-range dependencies between layers [6]. The solution to this problem is, almost always, either not mentioned, or circumvented by constraining pruning into targeting only layers

that do not induce problems [11]. However, these constraints are expected to reduce the efficiency of pruning.

In this paper we propose a solution to reduce effectively the size of networks using structured pruning, that were applied a mask using structured pruning. Our method is generic, automatic and reliably produces an effectively pruned network. We demonstrate its ability to operate on networks of any complexity by applying it on both a standard classification network [6] on the ImageNet ILSVRC2012 dataset [17] and on a more complex semantic segmentation network [18] trained on CityScapes [2]. We show that our solution allows gains in energy consumption and inference time on embedded hardware such as the NVIDIA Jetson AGX Xavier embedded GPU, providing an actual estimate of how structured pruning can be leveraged to reduce energy and latency footprints on a real hardware target.

II. RELATED WORKS

Originally designed to improve generalization of neural networks [10], pruning is now a popular method to reduce their memory or computational footprints. The most basic form of pruning involves masking out weights of least magnitude in a non-structured way [5]. This method does not reduce the size of the parameters’ tensors, but instead the introduced zeroes help compressing the network weights through encoded schemes [4]. However, getting any type of speed-up out of this method is difficult on most hardware [13].

To better leverage pruning on hardware, many methods instead apply “structured pruning”, that usually involves pruning whole neurons, *i.e.* filters in the case of convolution layers [11]. Other types of structured pruning exist, such as “filter shape pruning” [21] and this is why we will favor the “filter pruning” denomination to avoid ambiguity. Weight pruning and filter pruning are the two most popular types of pruning structures.

When pruning any type of structure, two aspects have to be tackled: 1) how to identify elements to prune and 2) how to prune them. The first issue can be solved using various types of pruning criteria. In the case of non-structured pruning, the magnitude of weights [5] or their gradient [15] are two popular criteria. When pruning filters, these criteria can be extended to either their norm over a filter [11] or a proxy that accounts for the whole filter’s importance, for example the multiplicative

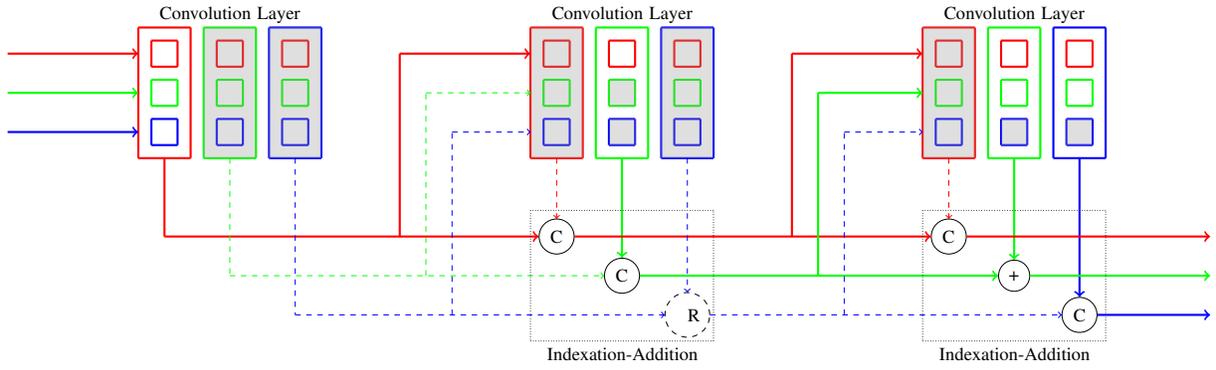


Fig. 1: Illustration of the difficulties when pruning filters in convolutional neural networks. Convolution layers are made of filters, each one outputting a channel (or “feature map”). Greyed out elements symbolise pruned filters and the kernels to remove to fit the dimensions of inputs (Problem 1). At the end of every residual block, the output of the last layer is summed with the input of the block. If the two tensors are pruned differently (Problem 3), what was an addition is now a mixture of additions (+), concatenations (C) or bypasses (dashed circles) that we call the *indexation-addition* operator (Section III-D). The consequence is that the final number of channels cannot be predicted solely from a particular layer in the network, but must be deduced by taking into account all the dependencies (Problem 2).

learned weight included in batch-normalization layers [12]. These criteria can be applied in two different ways: either they are used to identify the same (or a pre-determined) amount of weights/filters to remove in all layers (local pruning) or the target is set globally and the criterion is applied to all layers at the same time (global pruning).

Concerning the second issue, many popular methods apply a simple framework [4]: training the network, pruning a given proportion of weights by masking them away, fine-tuning the network and repeating the last two steps multiple times until a target pruning rate is reached. Other methods can involve a more progressive approach [7] that can include a regrowing mechanism [14]. Some techniques propose a more continuous way to prune weights, for example by applying them a penalty during training [20].

III. METHOD

A. Consequences of Structured Pruning

In Section II, we explained what is structured pruning. In order to present the problems it can induce, as well as the solutions we propose, we need to introduce some notations.

Let \mathcal{N} be a convolutional neural network. For the sake of convenience, we will consider that it is only made of convolutional layers l^i , whose input and output dimensions are f_{in}^i and f_{out}^i . Each convolution contains $f_{out}^i \times f_{in}^i \times k_h^i \times k_w^i$ weights \mathbf{w}^i (with $k_h^i \times k_w^i$ the size of the layer’s kernel) and f_{out}^i biases \mathbf{b}^i . A filter corresponds to the $f_{in}^i \times k_h^i \times k_w^i$ weights and one bias that produce one of the f_{out}^i channels in the output feature maps. Each of these layers operates on feature maps of size $f_{in}^i \times h^i \times w^i$ with $h^i \times w^i$ the resolution of the feature maps. In the case of networks such as ResNet [6] or HRNet [18], different layers can take the same feature maps as an input and multiple feature maps can be summed together. This simplified presentation is sufficient to expose the problems induced by global pruning.

a) *Problem 1*: Pruning filters reduces the output dimension f_{out} of a layer. Therefore, the dimension of its output is different and the input dimension f_{in} of the following layers must be adapted. This problem is well-known in the literature [11] and easy to solve in simple networks.

b) *Problem 2*: Residual connections [6] can introduce long-range dependencies and, therefore, identifying all the layers impacted by the change in dimension can be difficult. This problem is usually solved by avoiding pruning layers involved in such dependencies [11], but this solution is suboptimal.

c) *Problem 3*: Residual connections [6] usually involve summing together feature maps, that must therefore be of same dimensions, which is not the case anymore after global pruning. In the case of local pruning, feature maps are of the same dimensions, but the same mask may not have been applied on both feature maps, and summing together channels that are meant to be summed together produces a tensor of higher dimensions. This problem is less discussed in the literature and mostly solved using custom operations to manually adapt dimensions of feature maps [8].

These three problems, illustrated in Figure 1, are either eluded or not solved in the literature, even though most papers deal with ResNet-based architectures that are causing all of the three. Some expertise allows manually figuring out dependencies in such networks, but the complexity can get out of hand in the case of networks such as HRNets [18]. Indeed, missing any of these problems makes the networks impossible either to run efficiently or to run at all on hardware. This is the reason why we propose a method that can automatically and reliably produce pruned networks that can be ran efficiently on hardware.

B. Generalizing Operators to Handle a Subset of Channels

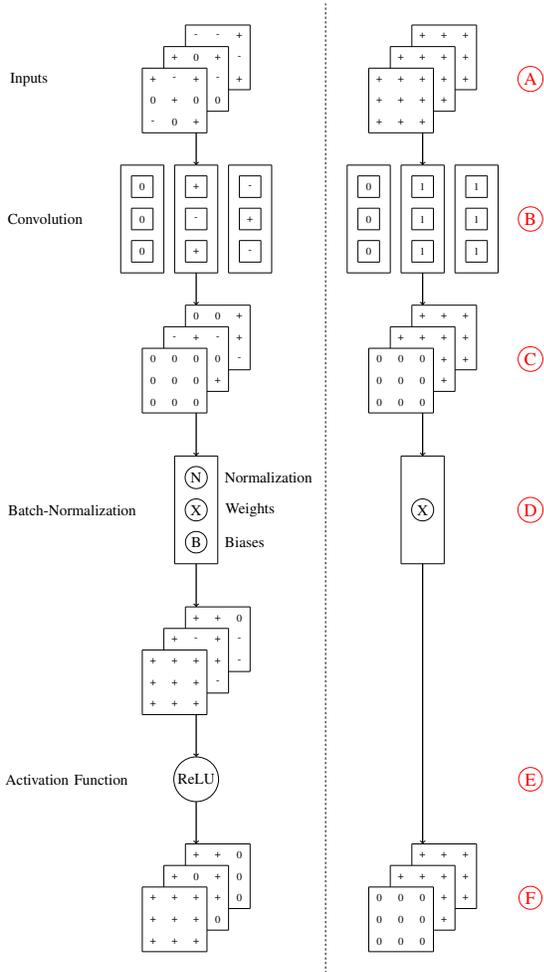


Fig. 2: Illustration of the proposed method to identify disconnected weights, with the original network on the left and the modified version on the right. (A) Input tensors are uniform to avoid unwanted null values, (B) weights of layers are replaced with their mask, therefore (C) the output only contain null values if a filter is pruned. (D) Normalization and biases are removed to keep null values null and (E) activation functions are removed not to add extra ones. The final output (F) allows deducing which filters are pruned.

The first step of our method is to make sure a given network is robust to pruning. Indeed, networks such as ResNets or HRNets contain operations that are applied to outputs of multiple layers. In such cases, the involved tensors must be of the same dimension, which may not be the case anymore after pruning. In the case of ResNets and HRNets, all operations of these types are additions of two tensors, such as those at the end of every residual connection. This means that we can tackle this problem by replacing additions with a generalized operator able to handle missing filters in any of its inputs.

To this mean, we replace additions with a new *indexation-addition* operation, with \mathbf{a} and \mathbf{b} the tensors to sum, that contain respectively n^a and n^b channels, \mathbf{i}^a and \mathbf{i}^b two lists

of indices and the output tensor \mathbf{c} , that contains n^c channels, defined in Equation (1):

$$\forall k \in \llbracket 1; n^c \rrbracket, \mathbf{c}_k = \begin{cases} \mathbf{a}_{\mathbf{i}_k^a}, & \text{if } \mathbf{i}_k^a \in \llbracket 1; n^a \rrbracket \\ \emptyset, & \text{otherwise} \end{cases} + \begin{cases} \mathbf{b}_{\mathbf{i}_k^b}, & \text{if } \mathbf{i}_k^b \in \llbracket 1; n^b \rrbracket \\ \emptyset, & \text{otherwise} \end{cases} \quad (1)$$

If $n^a = n^b$, $\mathbf{i}^a = [1, 2, \dots, n^a]$ and $\mathbf{i}^b = [1, 2, \dots, n^b]$, this *indexation-addition* operation is purely equivalent to an element-wise addition. Properly parameterized by adequate \mathbf{i}^a and \mathbf{i}^b , this operation allows leveraging any type of filter pruning. It is however necessary to find the right \mathbf{i}^a and \mathbf{i}^b and we provide a solution in Section III-D. Figure 1 illustrates how our solution relates to the problems mentioned in Section III-A and provides a simple way to view how it can behave like a mix of additions and concatenations.

C. Automatic Adaptation of Networks

Once the network is prepared for pruning by the introduction of this new *indexation-addition* operation to fit any distribution of the sparsity induced by pruning, the next step of the method is to identify automatically all dependencies between filters, kernels, biases or any sort of weights in the network. In summary, it is necessary to search for all the parts of the network that are disconnected when removing filters.

To identify all parameters whose contribution in a network's function is null, one can use its gradient over, for example, a mini-batch from the training data. Indeed, provided this mini-batch is a satisfying approximation of the network's domain of definition, a null gradient means that the network's function is null relatively to the involved weights, or at least constant in the case of biases. However, for our use-case, this is insufficient: not only does it not allow removing disconnected biases that still produce constant outputs that somehow contribute to the function, but it may also identify some isolated weights as pruned in a non-structured way, while it is not possible to leverage them.

This is why we instead operate on an architectural abstraction of the network, which is a copy of it that received three modifications that are illustrated in Figure 2:

- Its biases are removed to prevent them from adding a constant output that makes some disconnected/useless weights downstream have a non-null gradient.
- Its activation functions, and other non-linear operations such as normalization, are removed, so that a non-null input of a layer cannot produce a null output and gradient.
- The value of its weights are replaced by the value of the mask, made either of zeros or ones, so that, when fed with an input filled with non-null values of the same sign, the output cannot contain null values if it is not because of null, masked out weights.

Because of these modifications, a single input filled with non-null values of the same sign is enough to identify all

disconnected weights. Indeed, this network behaves like a purely linear and positive function and any null gradient in its parameters can only be due to a null function that can be removed. Weights, identified as disconnected in this copy network, are then removed from the original network.

D. Automatic Indexation

To deduce automatically the right \mathbf{i}^a and \mathbf{i}^b defined in Section III-B, we add another modification to the copy network described in Section III-C: we apply an *identity convolution* to the two tensors before summing them together. This *identity convolution* has weights of shape $n \times n \times 1 \times 1$ (with n the number of channels in the input tensor) whose values equates that of an identity matrix.

The gradient of the weights of this *identity convolution* allows deducing the corresponding list of indices. Indeed, once the null rows and columns of its weights are removed, the output dimensions are the same for both tensors to be summed while the input dimension matches that of the input tensors after pruning. The zero and non-zero remaining coefficient allows deducing how to map the input and output channels.

E. Summary of the Method

Here are all the steps to follow to apply our method:

Algorithm 1 Summary of the Method

- 1: train the network \mathcal{N}
 - 2: generate the pruning mask \mathbf{m} that masks out filters
 - 3: create a copy \mathcal{N}' of the network
 - 4: remove all biases \mathbf{b} from \mathcal{N}'
 - 5: remove all activation functions and normalization from \mathcal{N}'
 - 6: replace the weights \mathbf{w} of \mathcal{N}' by \mathbf{m}
 - 7: insert the *identity convolutions* where needed in \mathcal{N}'
 - 8: generate an input tensor \mathbf{x} , of adequate size, filled with ones and run $\mathcal{N}'(\mathbf{x})$
 - 9: compute $\frac{d\mathcal{N}'}{d\mathbf{w}}(\mathbf{x})$
 - 10: generate the new pruning mask \mathbf{m}' that masks away all weights whose gradient is null in \mathcal{N}'
 - 11: apply \mathbf{m}' to \mathcal{N} and mask away biases whose weights are pruned
 - 12: deduce from the mask of the *identity convolutions* the right \mathbf{i}^a and \mathbf{i}^b to replace additions with *indexation-addition* operations where needed
-

The method, summed up in Algorithm 1, solves all problems presented in Section III-A. It allows pruning a network and then generating its nearest equivalent whose dimensions are consistent and that can be leveraged on hardware. Since our method not only removes weights of null contribution but also biases whose gradient is constant, the function of the network is not preserved. However, the impact on accuracy is negligible and detailed in our experiments in Section IV-B.

IV. EXPERIMENTS

In our experiments, we will first detail the impact of our method on both the accuracy of the network and the

evaluation of its compression rate. Then we will demonstrate how the networks, whose type of sparsity usually prevents running inference, can be leveraged efficiently on resource-limited hardware. Our source code is available at: <https://github.com/HugoTessier-lab/Neural-Network-Shrinking.git>

A. Training conditions

a) *ImageNet*: We trained ResNet-50 [6] on the ImageNet ILSVRC2012 image classification dataset [17] for 90 epochs with a batch-size of 170 and a learning rate of 0.01 reduced by 10 every 30 epochs. We used the SGD optimizer with weight decay set to $1 \cdot 10^{-4}$ and momentum set to 0.9.

b) *Cityscapes*: We trained the HRNet-48 network [18] on the Cityscapes semantic segmentation dataset [2] for 200 epochs with a batch size of 10 and a learning rate of 0.01 reduced by $(1 - \frac{\text{current_epoch}}{\text{epochs}})^2$ at each epoch. We used the RMI loss [22] and the SGD optimizer with weight decay set to $5 \cdot 10^{-4}$ and momentum set to 0.9. During training, images are randomly cropped and resized, with a scale of $[0.5, 2]$, to $3 \times 512 \times 1024$. Data augmentation involves random flips, random Gaussian blur and color jittering.

c) *Pruning*: We prune networks following the method of Liu et al. [12]: pruning is divided in three iterations, with a linearly growing proportion of removed filters until the final pruning rate is matched. At each iteration, filters are masked out depending on the magnitude of the weight of their batch-normalization layer. After each iteration, ResNet-50 fine-tuned during 10 epochs and HRNet-48 during 20 epochs. The method of Liu et al. [12] also implies penalizing weights of batch-normalization layers with a smooth- \mathcal{L}_1 norm, with an importance factor of $\lambda = 10^{-5}$ for ResNet-50 and $\lambda = 10^{-6}$ for HRNet-48.

B. Impact on Accuracy and Compression Rate

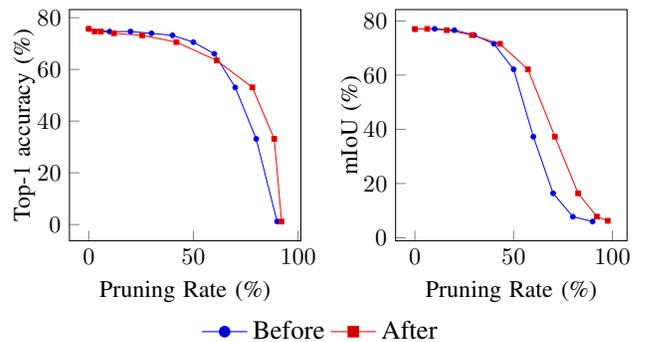


Fig. 3: For ResNet-50 on ImageNet (left) or HRNet-48 on Cityscapes (right): accuracy depending on pruning rate, either in terms of proportion of pruned filters (blue) or remaining parameters after application of our method (red).

In our experiments, we reported mostly no difference in accuracy before and after applying our method, as it can be seen in Figure 3. This implies that the parameters removed by

our method, that did not have a null contribution to the function, such as the remaining biases mentioned in Section III-C, might have had a negligible impact on the network’s accuracy. The only outliers are points where accuracy is already severely decreased, for example the accuracy of ResNet-50 pruned at 60% that goes from 66.05% to 63.478%, while the baseline is at 75.7%.

In Figure 3 we also show the trade-off between accuracy and two types of pruning rate: one defined as the proportion of removed filters, which is a widespread target criterion in the literature, and one defined as the exact count of remaining parameters in the network once our method has been applied. We see that using the percentage of removed filter is not faithful to the actual compression rate of the network. The actual trade-off is more advantageous once our method has been applied to both purge the network from useless weights and get a faithful estimation of all eliminated weights.

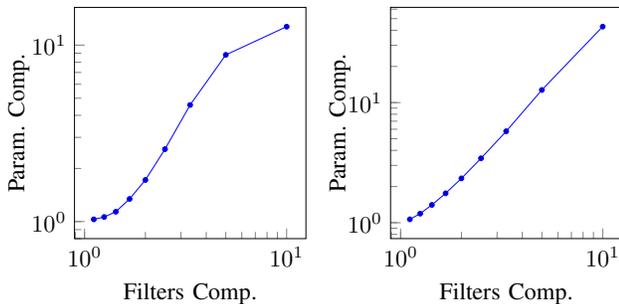


Fig. 4: For ResNet-50 on ImageNet (left) or HRNet-48 on Cityscapes (right): relation between the estimated compression rate in terms of pruned filters (x-axis) and remaining parameters after reducing the network using our method (y-axis).

In Figure 4, we compare the compression rate (*i.e.* $\frac{100\% - \text{pruning_rate}\%}{100\%}$) in terms of removed filters or removed parameters, *i.e.* before and after our method. The relationship between the two measures seem to depend on the involved architecture and we expect it to depend on the pruning criterion too.

C. Impact on Hardware

To measure the inference time and energetic consumption of pruned networks on NVIDIA Jetson AGX Xavier in the “30W All” mode, we first converted our networks to ONNX, that is a format that can be handled by many frameworks on most hardware. The *indexation-addition* operations were implemented using *ScatterND* and *transpose* operators. *ScatterND* allows operating on slices in tensors and transpositions allow operating specifically on channels, while *Scatter* is element-wise and requires storing a cumbersome array of indices. Before summation, both tensors need to be scattered into a temporary tensor, that is instantiated dynamically. We used the JetPack SDK 5.0, with CUDA 11.4.14, cuDNN 8.3.2, TensorRT 8.4.0 EA and ONNX Runtime 1.12.0. Energetic consumption was given using the *tegrastats* utility. Inference on ResNet-50 is run with an input of size $(1 \times 3 \times 224 \times 224)$

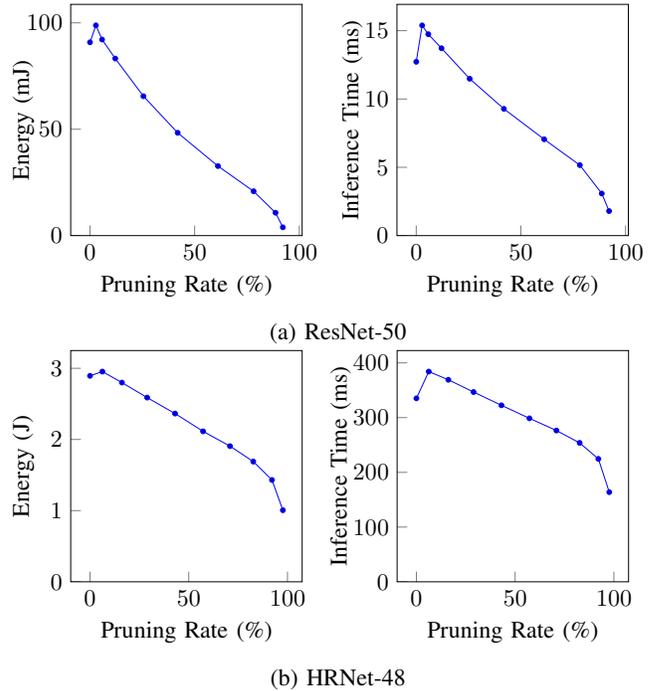


Fig. 5: Energetic consumption and inference time of ResNet-50 and HRNet-48, depending on the pruning rate in terms of parameters, on NVIDIA Jetson AGX Xavier in the “30W All” mode, using JetPack SDK 5.0 and ONNX Runtime 1.12.0 running with the TensorRT execution provider. Results are averaged over 10k inferences with inputs of size $(1 \times 3 \times 224 \times 224)$ after 1k runs of warm-up for ResNet-50 and 1k inferences with inputs of size $(1 \times 3 \times 512 \times 1024)$ after 100 runs of warm-up for HRNet-48.

and HRNet-48 with one of size $(1 \times 3 \times 512 \times 1024)$. ONNX Runtime was used with the TensorRT execution provider, that turned out to be the one that gave the best inference time.

Figure 5 provides results for ResNet-50 on ImageNet and HRNet-48 on Cityscapes. Both show similar tendencies: at first, the extra cost of *indexation-addition* operations takes a toll on the efficiency of pruning, but after that initial jump, the cost of networks, either in terms of energy consumption or inference time, decreases significantly. This shows that, although a better implementation of the *indexation-addition* operations would be beneficial, our current solution is enough for free and unconstrained structured pruning to be cost effective. Therefore, we can say that it is possible to leverage efficiently any type of filter pruning in even complex deep convolutional neural networks.

V. DISCUSSION

Three observations can be drawn from our experiments: 1) our method allows a more reliable measurement of the count of remaining parameters in the network, as can be seen in Figure 3, 2) the relation between this accurate pruning rate and inference time or energy consumption is non-linear and 3) the cost introduced by our custom operators is not negligible and

makes the least pruned networks cost more than non-pruned ones, as can be seen in Figure 5.

The first two observations show that our method is a useful tool to better study the efficiency of unconstrained filter pruning. Indeed, it produces a network in which the vast majority of remaining parameters are guaranteed to contribute to the function, with the marginal exception of some isolated weights that may be inactive by accident. Therefore, it is now possible to directly measure the accuracy-to-energy or accuracy-to-latency trade-off, which provide a more relevant insight into the impact of pruning on hardware than a more theoretical accuracy-to-parameters trade-off. Since this is not the focus of this article, we did not provide such an analysis and did not choose the pruning method that gave the absolute best possible performance. This will be the focus of future contributions. This ability to provide a more faithful compression rate than the naive rate of removed filters also allows better controlling the growth of pruning rate between pruning iterations. This is likely to help improving performance and avoiding to remove entire layers by accident, which is called *layer collapse* [19].

Concerning the last observation, finding the best implementation of the custom operators, necessary to run pruned networks, obviously requires further investigation. Using `trtexec`, we did the profiling of the operators of the HRNet-48, with 10% of the filters pruned and 6.26% of removed parameters, which is the HRNet-48 with the highest inference time. It turned out that the “Foreign Nodes” generated by TensorRT, that contain the *ScatterND* we used for our *indexation-addition* operations, are responsible for 14.8% of the total inference time. When subtracting the cost of these nodes from the network’s total average time of 369.8ms according to `trtexec`, the remaining inference time is of 314.3ms, which is actually lower than that of the non-pruned network, which is of 318.7ms. This means that if an optimized implementation of the operators allowed their cost to be negligible, it would make pruning a lot more beneficial, even at low pruning rates.

VI. CONCLUSION

We have proposed an efficient and generic way to leverage any type of filter pruning in deep convolutional neural networks. Indeed, even though removing filters in a network can trigger a certain array of problems that can even prevent running its inference, our solution is able to tackle them and generates functional pruned networks that can be run efficiently on hardware. Our experiments, even though they show that our current ONNX implementation has a non-negligible cost, demonstrate that unconstrained filter pruning can be cost-effective.

REFERENCES

- [1] Chun-Fu Chen, Gwo Giun Lee, Vincent Sritapan, and Ching-Yung Lin. Deep convolutional neural network on ios mobile devices. In *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 130–135. IEEE, 2016.
- [2] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.
- [3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.
- [4] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [5] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.
- [6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [7] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.
- [8] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.
- [9] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.
- [10] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.
- [11] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.
- [12] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.
- [13] Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, Xuehai Qian, et al. Non-structured dnn weight pruning—is it beneficial in any platform? *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [14] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.
- [15] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.
- [16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [18] Ke Sun, Yang Zhao, Borui Jiang, Tianheng Cheng, Bin Xiao, Dong Liu, Yadong Mu, Xinggang Wang, Wenyu Liu, and Jingdong Wang. High-resolution representations for labeling pixels and regions. *arXiv preprint arXiv:1904.04514*, 2019.
- [19] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.
- [20] Hugo Tessier, Vincent Gripon, Matthieu Léonardon, Matthieu Arzel, Thomas Hannagan, and David Bertrand. Rethinking weight decay for efficient neural network pruning. *Journal of Imaging*, 8(3):64, 2022.
- [21] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.
- [22] Shuai Zhao, Yang Wang, Zheng Yang, and Deng Cai. Region mutual information loss for semantic segmentation. *Advances in Neural Information Processing Systems*, 32, 2019.