# Investigating the Not-So-Obvious Effects of Structured Pruning

Hugo Tessier, Vincent Gripon, Mathieu Léonardon, Matthieu Arzel, David Bertrand, Thomas Hannagan

# Investigating the Not-So-Obvious Effects of Structured Pruning

**Hugo Tessier** [1 2] **Vincent Gripon** [2] **Mathieu Léonardon** [2] **Matthieu Arzel** [2]
**David Bertrand** [1] **Thomas Hannagan** [1]

## Abstract

Structured pruning is a popular method to reduce the cost of convolutional neural networks. However, depending on the architecture, pruning introduces dimensional discrepancies which prevent the actual reduction of pruned networks and mask their true complexity. Most papers in the literature overlook these issues. We propose a method that systematically solves them and generate an operational network. We show through experiments the gap between the theoretical pruning ratio and the actual complexity revealed by our method.

## 1. Introduction

Deep convolutional neural networks are at the state of the art in many domains, such as computer vision. 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 (Chen et al., 2016). The field of deep neural networks compression counts multiple types of method, such as quantization (Courbariaux et al., 2015) or distillation (Hinton et al., 2015). The one we focus on in this article is pruning (Han et al., 2015b), that involves removing unnecessary weights from a network.

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, which does not reduce the cost of networks. Leveraging pruning to get gains on hardware is actually not a trivial task. Pruning isolated weights (Han et al., 2015b) ("non-structured pruning") produces sparse matrices, that are difficult to accelerate (Ma et al., 2021). 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 (He et al., 2016).

In this paper we propose a solution to reduce effectively the size of networks 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 (He et al., 2016) on the ImageNet ILSVRC2012 dataset (Russakovsky et al., 2015) and on a more complex semantic segmentation network (Sun et al., 2019) trained on CityScapes (Cordts et al., 2016). Our experiments show that not taking into account the aforementioned problems tends to distort the apparent trade-off between the computational complexity and accuracy, which harms the relevance of the results. Therefore, our method is a useful tool for a more reliable study of pruning.

## 2. Related Works

When pruning a network, three aspects have to be tackled: 1) what kinds of parts to prune, 2) how to identify those to prune and 3) to prune them. We already mentioned how the first aspect can be divided into non-structured and structured pruning. The second issue can be solved using various types of pruning criteria. In the case of non-structured pruning, the magnitude of weights (Han et al., 2015b) or their gradients (Molchanov et al., 2016) are two popular criteria. In the case of structured pruning, these criteria can be extended to either their norm over a filter (Li et al., 2016) or a proxy that accounts for the whole filter's importance, for example the multiplicative learned weight included in batch-normalization layers (Liu et al., 2017). 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, possibly unevenly, at the same time (global pruning).

Concerning the third issue, many popular methods apply a simple framework (Han et al., 2015a): 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 (He

[1]*Stellantis*, Vélizy-Villacoublay, France [2]*IMT Atlantique*, Lab-STICC, UMR CNRS 6285, F-29238 Brest, France. Correspondence to: Hugo Tessier <hugo.tessier@imt-atlantique.fr>.

*Proceedings of the $39^{th}$ International Conference on Machine Learning*, Baltimore, Maryland, USA, PMLR 162, 2022. 

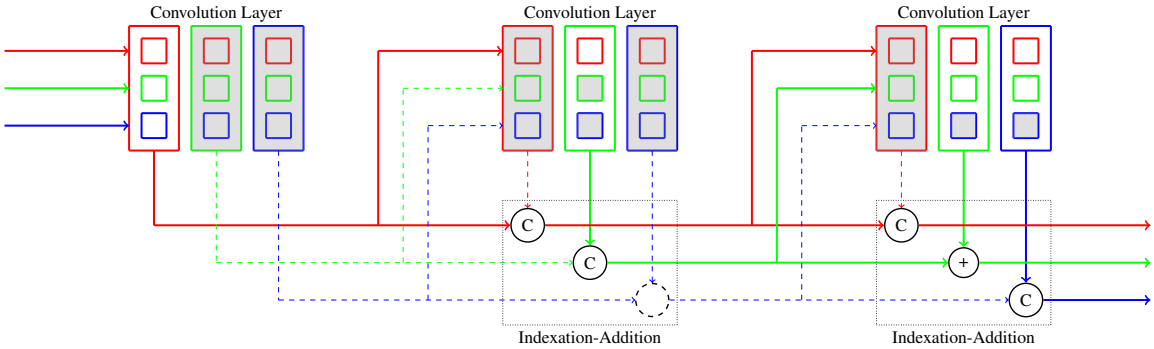


*Figure 1.* Pruning filters (greyed-out columns) means producing fewer output channels: the input dimensions of the following layers must be adapted (greyed out rows). However, the number of channels a layer gets as an input is not easy to predict: because of the addition at the end of each residual block, the respective index of each channel must be conserved in the destination tensor. Summing together the outputs of two pruned layers can produce a tensor whose number of channels is different from that of any of the two summed tensors, as the addition must be replaced with a mix of concatenations and additions (which we call an *indexation-addition* operation). Therefore, the dimensions of tensors and of the layers can only be deduced by taking into account all dependencies in the network at once.

et al., 2018) that can include a regrowing mechanism (Mocanu et al., 2018). Some techniques propose a more continuous way to prune weights, for example the application of a penalty on them during training (Tessier et al., 2022).

## 3. Method

In Section 2, we explained what is structured pruning. Pruning filters can introduce some dimensional discrepancies, especially in architectures such as ResNets (He et al., 2016), that must be tackled to be able to allow proper inference. Figure 1 and its caption sum up the different problems that can occur and that our method is designed to solve.

These problems are often overlooked in the literature, even though most papers deal with ResNet-based architectures that are subject to all of them. 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 (Sun et al., 2019). Missing any of these problems makes the networks impossible either to run efficiently or to run at all. Moreover, avoiding these problems by only putting pruned weights to zero, and thus not taking into account any of these dependencies, leads to evaluate incorrectly the actual runtime complexity of pruned networks, which harms the reliability of their study. This is the reason why we propose a method that can automatically and reliably produce pruned networks in which of all these problems have been solved.

### 3.1. Automatic Adaptation of Networks

The first step of our method involves identifying all the parts of the network that are disconnected when removing filters by taking into account all the dependencies.

To identify all the parameters whose contribution in a network's function is null, one can use their gradients 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, which we do not want to consider.

This is why we instead operate on an architectural abstraction of the network, which is a copy of it that received modifications that are illustrated and described in Figure 2 and its caption. These modifications allow a single input, filled with non-null values of the same sign, to be 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, effectively reducing its size.

### 3.2. Automatic Indexation in Additions

Deep convolutional networks can contain operations that create constraints on the tensors they take as an input. In the case of networks such as ResNets (He et al., 2016) or HRNets (Sun et al., 2019), the addition at the end of every residual block needs its two input tensors to be of the exact same dimensions, which may not be the case anymore after pruning. We can tackle this problem by replacing additions

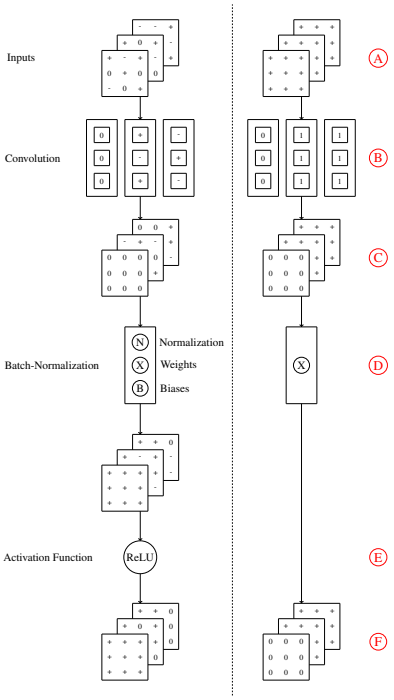


*Figure 2.* Illustration of the proposed method to identify disconnected weights, with the original network on the left and the modified version on the right. (A) Input tensors are filled with positive values to avoid unwanted zeros, (B) weights of layers are replaced with their mask, therefore (C) the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

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).

If $n^a = n^b$, $\mathbf{i}^a = [1, 2, \ldots, n^a]$ and $\mathbf{i}^b = [1, 2, \ldots, n^b]$, 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 3.2. Figure 1 illustrates how our solution relates to the problems mentioned in its caption and provides a simple way to view how it can behave like a mix of additions and concatenations.

$$\forall k \in [\![1; n^c]\!], \mathbf{c}_k = \begin{cases} \mathbf{a}_{\mathbf{i}^a_k}, & \text{if } \mathbf{i}^a_k \in [\![1; n^a]\!] \\ \varnothing, & \text{otherwise} \end{cases} + \begin{cases} \mathbf{b}_{\mathbf{i}^b_k}, & \text{if } \mathbf{i}^b_k \in [\![1; n^b]\!] \\ \varnothing, & \text{otherwise} \end{cases} \quad (1)$$

To deduce automatically the right $\mathbf{i}^a$ and $\mathbf{i}^b$, we add another modification to the copy network described in Section 3.1: 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, $1 \times 1$ because we consider 2d inputs) 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 coefficients allow deducing how to map the input to the corresponding output channels.

### 3.3. Summary

Algorithm 1 sums up our overall 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 the biases $\mathbf{b}$ from $\mathcal{N}'$
5: remove all the activation functions and normalizations 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 $\frac{\delta \mathcal{N}'}{\delta \mathbf{w}}(\mathbf{x})$
10: generate the new pruning mask $\mathbf{m}'$ that masks away all the weights whose gradients are 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

This method produces pruned networks in which interdependant weights are guaranteed to be pruned altogether, which prevents any dimensional mismatch between layers. Our method also adapts the addition operations so that they can accommodate tensors of different number of channels. Therefore, all the filters and kernels are guaranteed to contribute to the network's function, which allows a more reliable measurement of the actual pruning rate and associated complexity of pruned networks.

## 4. Experiments

In our experiments, we will detail the impact of our method on both the accuracy of the network and the evaluation of its compression rate. Our source code is available at: https://github.com/HugoTessier-lab/Neural-Network-Shrinking.git.

### 4.1. Training conditions

**ImageNet** We trained ResNet-50 (He et al., 2016) on the ImageNet ILSVRC2012 image classification dataset (Russakovsky et al., 2015) 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.

**Cityscapes** We trained the HRNet-48 network (Sun et al., 2019) on the Cityscapes semantic segmentation dataset (Cordts et al., 2016) for 200 epochs with a batch size of 10 and a learning rate of 0.01 reduced by $(1 - \frac{current\_epoch}{epochs})^2$ at each epoch. We used the RMI loss (Zhao et al., 2019) 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.

**Pruning** We prune networks following the method of Liu et al. (Liu et al., 2017): 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. (Liu et al., 2017) 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.

### 4.2. Impact on Accuracy and Compression Rate

In our experiments, we mostly reported no difference in accuracy before and after applying our method, as it can be seen horizontally 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 3.1, 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 66.1% to 63.5%, while the baseline is at 75.7%.

Figure 3 features another, more significant, difference between the two curves, as they show the trade-off between accuracy and two different types of pruning rate: one defined as the proportion of removed filters, which the target criterion, widespread in the literature, that we used when pruning the networks, and one defined as the exact count of remaining parameters in the network once our method has been applied after pruning. We see that using the percentage of removed filters is not a reliable indicator of 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 reliable estimation of all eliminated weights.

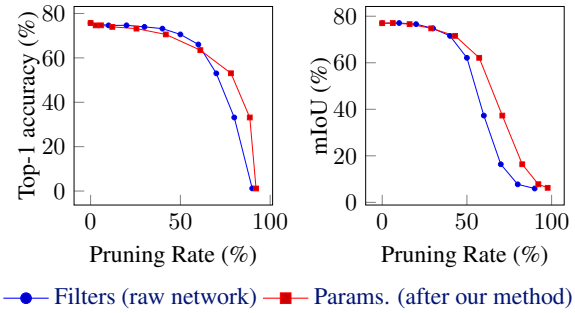


*Figure 3.* For ResNet-50 on ImageNet (left) or HRNet-48 on Cityscapes (right): accuracy depending on pruning rate, either in terms of proportion of pruned filters (blue) or remaining parameters after application of our method (red).

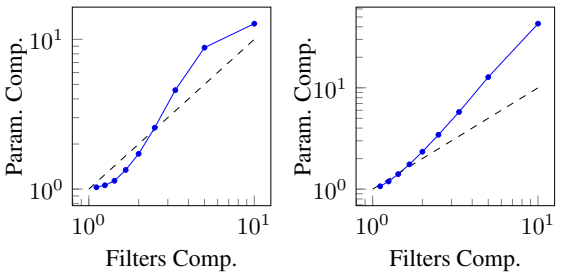


*Figure 4.* For ResNet-50 on ImageNet (left) or HRNet-48 on Cityscapes (right): relationship between the estimated compression rate in terms of pruned filters (x-axis) and remaining parameters after reducing the network using our method (y-axis). The dashed lines provide a $y = x$ reference.

In Figure 4, we compare the compression rate (*i.e.* $\frac{100\%}{100\% - pruning\_rate\%}$) in terms of removed filters or removed parameters, *i.e.* before and after our method. The relationship between the two measures seems to differ greatly depending on the involved architecture, and we expect it to depend on the pruning criterion too. Moreover, in either cases, the relationship is non-linear, which highlights that the rate of remaining parameters in our operational models is hardly predictable from the initial rate of pruned filters.

## 5. 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 induce a certain array of problems that can even prevent inference, our solution is able to tackle them and generate operational pruned networks that can be used for a more reliable study of the impact of pruning.

## References


Chen, C.-F., Lee, G. G., Sritapan, V., and Lin, C.-Y. Deep convolutional neural network on ios mobile devices. In *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 130–135. IEEE, 2016.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

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

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015b.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

Hinton, G., Vinyals, O., Dean, J., et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

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

Ma, X., Lin, S., Ye, S., He, Z., Zhang, L., Yuan, G., Tan, S. H., Li, Z., Fan, D., Qian, X., et al. Non-structured dnn weight pruning–is it beneficial in any platform? *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9 (1):1–12, 2018.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Sun, K., Zhao, Y., Jiang, B., Cheng, T., Xiao, B., Liu, D., Mu, Y., Wang, X., Liu, W., and Wang, J. High-resolution representations for labeling pixels and regions. *arXiv preprint arXiv:1904.04514*, 2019.

Tessier, H., Gripon, V., Léonardon, M., Arzel, M., Hannagan, T., and Bertrand, D. Rethinking weight decay for efficient neural network pruning. *Journal of Imaging*, 8 (3):64, 2022.

Zhao, S., Wang, Y., Yang, Z., and Cai, D. Region mutual information loss for semantic segmentation. *Advances in Neural Information Processing Systems*, 32, 2019.