A Relevance-Based CNN Trimming Method for Low-Resources Embedded Vision

Dalila Ressi[1 [0000−0001−5291−5438]], Mara Pistellato[1 [0000−0001−6273−290X]], Andrea Albarelli[1 [0000−0002−3659−5099]], and Filippo Bergamasco[1 [0000−0001−6668−1556]]
[1]DAIS, Università Ca'Foscari Venezia, 155, via Torino, Venezia Italy
{dalila.ressi, mara.pistellato, albarelli, filippo.bergamasco}@unive.it

Abstract. A significant amount of Deep Learning research deals with the reduction of network complexity. In most scenarios the preservation of very high performance has priority over size reduction. However, when dealing with embedded systems, the limited amount of resources forces a switch in perspective. In fact, being able to dramatically reduce complexity could be a stronger requisite for overall feasibility than excellent performance. In this paper we propose a simple to implement yet effective method to largely reduce the size of Convolutional Neural Networks with minimal impact on their performance. The key idea is to assess the relevance of each kernel with respect to a representative dataset by computing the output of its activation function and to trim them accordingly. The resulting network becomes small enough to be adopted on embedded hardware, such as smart cameras or lightweight edge processing units. In order to assess the capability of our method with respect to real-world scenarios, we adopted it to shrink two different pre-trained networks to be hosted on general purpose low-end FPGA hardware to be found in embedded cameras. Our experiments demonstrated both the overall feasibility of the method and its superior performance when compared with similar size-reducing techniques introduced in recent literature.

Keywords: Computer Vision · CNN · Industrial Application · Compression · Filter Pruning

1 Introduction

Over the past few years industrial applications have been exploiting extensively the advantages coming from Deep Learning, in particular when using Convolutional Neural Networks (CNNs). The intrinsic flexibility of these networks makes them widely adopted in a variety of practical applications, from medical to industrial. In particular, image and signal processing tasks are well-suited for the convolutional architecture, therefore a huge number of Computer Vision solutions have been proposed in the literature. Deploying a large and accurate model to perform a certain task takes considerable energy and space. Even if this might not be a problem during the training phase, it becomes a big issue at inference

time, especially if the model has to run on devices with reduced computational resources or small storage space.

This is the case with many modern applications, including IoT devices, smart cameras, drones, smartphones or any other kind of device characterized by limited resources and low energy consumption requirements. For this reason, the adaptation of inference networks to embedded systems has been covered by many researchers [27, 5], devising solutions ranging from architectures specially crafted for Field Programmable Gate Arrays (FPGAs) [23, 13] to techniques focused on low consumption for wireless and mobile devices [28]. Indeed, according to the type of task to perform, two main approaches are to be found in literature. The choice is between training from scratch a smaller specialized network, or compressing a large pre-trained network and adapting it for the task (pruning).

In this paper we introduce a pruning technique that, while of general application, has been developed to address a specific vision task. Namely, our goal was to synthesize on FPGA hardware, available on commercial smart cameras, a lightweight CNN to locate both 1D and 2D signal peaks respectively in line scan and area scan images. To this end, we started from full-size networks and we made them tiny by means of a novel pruning algorithm which does not require manual tuning of parameters and allows to greatly reduce the number of floating point operations (FLOPs) computed. In a throughout experimental section we show that the trimmed network achieves better results with respect to a network with the pruned architecture that is trained starting from random weights. Moreover, the resulting performance is better than the one obtainable with other state-of-the-art trimming methods. Finally, by using other compression techniques such as quantization of the weights [3, 24] the resulting model can even be further reduced.

2 Related work

Fitting large inference networks to embedded systems is a topic that attracted the attention of many researchers in the recent past. A typical solution consists to re-design network components to achieve similar results with smaller resources. It is the case of MobileNets [10] and ThinNet [2], where the authors substitute classical convolutions with depthwise separable convolutions. Another example is SqueezeNet [12], achieving the same accuracy of AlexNet with 50 times less parameters. It uses multiple strategies to reduce each layer complexity, like design space exploration and the introduction of new modules. Such solutions are usually targeted for mobile devices [14, 29] but can also be implemented in large networks to reduce their size. Another interesting approach, called distillation [9], consists in using a larger network to teach the same task to a smaller network. However, it can only be used for classification.

Great effort has also been put on techniques to compress existing networks by reducing the number and/or precision of each weight. The former is usually referred as *Pruning* [1] and the latter as *Quantization* [3, 24, 6]. Pruning comes

in a lot of different flavours, and sometimes it is difficult to actually understand which method is the best [30] considering the high inconstancy of performance for different application scenarios [20]. Weight pruning techniques [7] remove single connections by setting some weights to zero, but the resulting sparse matrices cannot exploit BLAS libraries and are hard to implement on FPGA [6]. A simpler and more structured manner is to prune whole *kernels* from a Convolutional Neural Network. This procedure is often referred as *filter pruning* or *trimming* [11]. Our work belongs to this category.

As discussed in [17], filter pruning techniques can be further categorized into two groups: methods focusing on *property importance* and others concentrating on *adaptive importance*. In the first group we find methods which prune filters according to intrinsic properties of the networks, and do not modify the training loss. In 2016, Hu et al. [11] proposed a layer-wise method which analyses the neuron outputs to compute the *Average Percentage of Zero (APoZ)* activations after the ReLU mapping. The idea is to remove neurons with an APoZ larger than one standard deviation from the average APoZ of the target trimming layer. Instead of looking at the outputs, the method proposed by Li et al. [16] measures the relative importance of a filter in each layer by computing the sum of its absolute weights (i.e. its *L1-Norm*). A more recent approach has been suggested by He et al. [8]. It expands the norm-based filter criterion by computing the Geometric Median (*GM*) of the filters within the same layer. The idea is that filters close to the GM can be represented by the other filters, and therefore are good candidates to be pruned. The authors illustrate how the smallest norm filters can be very important, as they could actually be larger than zero or they can have a small norm deviation. In 2020 Lin et al. [17] proposed a method called Filter Pruning using High-Rank Feature Maps (*HRank*). They claim that average rank of multiple feature maps generated by a single filter is independent from the distribution of the images. Filters which generate lower-rank feature maps are less important and can be removed first in a one-shot manner, requiring only a few fine-tuning epochs after the pruning phase.

Adaptive importance methods like [19, 18] usually achieve better compression and speed-up than property importance based ones. On the other hand, these techniques change the loss function up to the point that retraining becomes a separated problem, usually requiring to search again a new best set of hyper-parameters.

There is a last class of filter pruning algorithms that deserves to be mentioned. Sometimes filter pruning is exploited to find the best sub-network from an original one. It is the case of [4] and [26] where they use PCA to compressing both length and width of the network. Our goal however is to compress an existing network without the need to retrain it completely. For this reason, we focus on *Property Importance* algorithms.

Methods aiming to remove parameters from the network, regardless where the connections or the filters are, can be considered *global*. Usually there is only one threshold to be set, such as the number of filters or the compression rate to achieve. Some methods focus on specific layers, usually relying on certain

statistics to pick the most promising ones. Rather than global methods, these *layer-wise* pruning algorithms require more than one threshold or other parameters to be set, making them less robust.

Finally, pruning is usually performed in three stages: (i) preparation of an appropriately large network either by training it from scratch or by adapting an existing trained network using transfer learning; (ii) removal of superfluous parameters and (iii) fine-tuning to recover the loss of accuracy. The second and third stages are often repeated until the network reaches the desired level of compression. Some methods, however, perform pruning in a *one-shot* fashion by removing a chosen set of weights in a single pass [17].

3 The pruning method

We propose a global filter pruning method based on the idea that kernels can be ranked by means of a *relevance metric* computed according to the output after the activation function. Less relevant filters are iteratively removed in a *prune one and re-train* fashion, to allow the network to adjust to the reduced channel. Conforming with this process, we call our method ReFT (Relevance-based Filter Trimming).

More formally, given a convolutional layer, its input can be represented as a 2D tensor $I \times H$ containing a signal of I single $H-$dimensional feature vectors[1]. The convolutional layer contains K different $S \times H$ kernels that are convolved with the input to produce an $I \times K$ output tensor. This operation requires approximately $S \times H$ multiply-add operations for I filter shift for a total of $SHIK$ operations. Our goal is to define a *relevance function* to be computed over the K kernels in order to iteratively remove the kernel with smaller relevance. Each kernel removal will result in a reduction of the multiply-add operation in the order of SHI. Furthermore, it will reduce the size of the output of the layer of a factor proportional to the input I.

In detail, the ReFT iterative reduction process is performed according to the pseudo-code described in Algorithm . Here, $Kernels$ is the full set of kernels in the network, $Kernels(i)$ is the subset of kernels at layer i and $Kernels(i, c)$ points to a specific kernel. The value $output(i, c, s)$ refers to the output of the activation function after $Kernels(i, c)$ for data point s.

The network reduction stops when either the number of kernels left is below a given threshold t or the performance of the network (measured by the function $perfMetric$) is less than minimum acceptable value ϵ. The actual characteristic measured by $perfMetric$ could change according to the application scenario. For instance it could be a function of the average loss with respect to a validation set or a specific metric over the confusion matrix.

During its main iteration the ReFT algorithm selects the least relevant kernel, removes it from the network and performs a partial retrain if the removed kernel was not negligible (i.e. exhibiting an output value of 0 for all the samples in $\mathcal{S}$.

[1] this is actually the case for 1D convolutional layers, but the extension to 2D layers is straightforward

Algorithm: ReFT network reduction

```
Choose a representative dataset 𝒮;
while |Kernels| > t and perfMetric > ϵ do
    for s ∈ 𝒮 do
        for i ∈ Layers do
            for c ∈ Kernels(i) do
                O_c^i(s) ← output(i, c, s)
            end
        end
    end
    (α, β) = arg min_{i,c} (relevance(i, c));
    Remove Kernels(α, β) from the network;
    if Σ_{s∈S} O_β^α(s) ≠ 0 then
        Retrain a few epochs to adapt model;
    end
end
```

Of course, key to the effectiveness of the ReFT reduction is the choice of a suitable *relevance* function. As anticipated when introducing the method, the main idea is to account for the output distribution of the kernel with respect to real data, rather than for its input weight (which is much more common in literature). The rationale is to look at the *actual* effect of the kernel, rather than at its *potential* impact expressed in an implicit way by its input weights. To this end, we define a *span* function as:

$$span(i, c, \gamma) = q_{100-\gamma}(O_c^i) - q_\gamma(O_c^i)$$

That is the difference between percentile $100 - \gamma$ and percentile γ in the distribution O_c^i over all the data points s in the dataset $\mathcal{S}$. Thus, we have that $span(i, c, 0)$ represents the distance between the maximum and the minimum output for $Kernels(i, c)$ over the dataset $\mathcal{S}$.

In principle, the *span* function could be directly adopted to define relevance by choosing a specific value for γ. However, in order to mitigate the effect of outliers, we would like to use the full span only to disambiguate between kernels with similar output distributions, while adopting more conservative percentiles most of the time.

To obtain this result, we define *relevance* as an implicit metric by defining this pairwise partial ordering function:

$$\begin{aligned} relevance(j,k) < relevance(l,m) \iff \\ span(j,k,2) < span(l,m,2) \quad \vee \\ span(j,k,2) = span(l,m,2) \quad \wedge \\ span(j,k,0) < span(l,m,0)) \end{aligned}$$

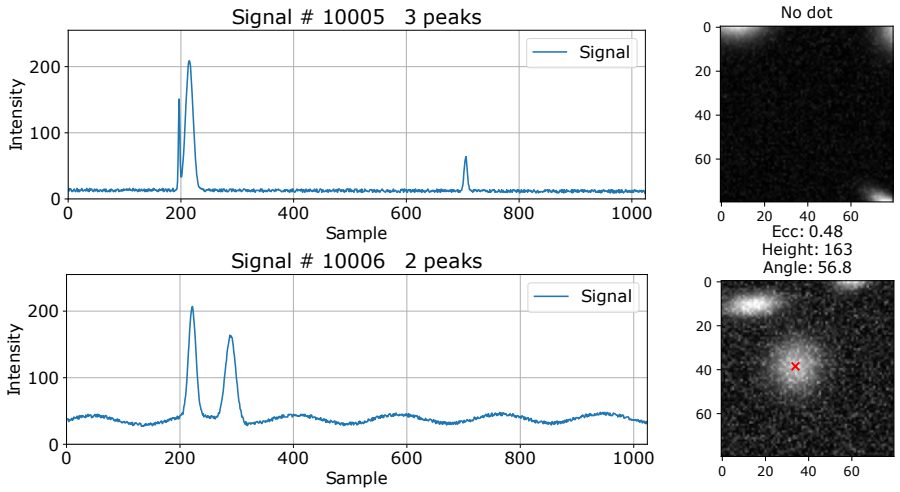


Fig. 1: Left: two scanlines acquired by the camera for the peak detection task. Multiple or no peaks can be present in the same scanline, sometimes very close to each other. Right: two samples from the *Dots* dataset. Also in this case there might not be any dot present or multiple ones. In the bottom right picture the detected dot center is highlighted by a red cross

In practice, this means that filter j, k is less relevant than filter n, m if the distance between the 98^{th} and the 2^{nd} percentile is lower or, if they are equal, the full span is lower. While it could seem counter-intuitive that the 98^{th} and the 2^{nd} can be frequently equal, this actually happens a lot due to the behaviour of clamping functions such as ReLU.

4 Applications and Experimental Evaluation

In this section we compare our method to other similar recent pruning algorithms introduced in §2. In order to have a fair comparison, we selected *property importance* methods focusing on the L1-norm of the filters [16], rank of the feature maps (HRank [8]), and layers' outputs like Apoz [11] and GM [17]. We first analyse the performances of the proposed technique (ReFT) for specific camera tasks, designed to be carried out in relatively small embedded devices. After that, we apply our pruning to VGG16 on Cifar10 to show that the proposed method is effective also when applied to more complex architectures.

4.1 Pruning CNNs for Camera Tasks

As already discussed, image processing for quality inspection in an industrial environment often involves strict requirements. For this reason, smart cameras

Model	Acc	Pos err	Height err	Params	Flops	Ratio
Peaks	0.999	0.342	0.0149	2.22K	2.412M	1.00
Peaks-ReFT	**0.993**	**0.644**	**0.0471**	0.936K	0.290M	0.42
Peaks-APoZ	0.988	0.717	0.0566	0.885K	0.295M	0.42
Peaks-L1-norm	0.984	0.806	0.0635	0.936K	0.290M	0.42
Peaks-GM	0.965	1.04	0.0784	0.936K	0.290M	0.42
Peaks-ReFT-S	0.989	1.1400	0.0578	0.936K	0.290M	0.42

Table 1: Average accuracy, average peak position error, average height error, number of parameters and compression ratio for Peak network with 34 pruned kernels (out of the 40 available distributed across 3 conv layers). Pruning methods have been repeated 20 times. ReFT-S is the training of Peak-ReFT with random weights. See fig.2 for standard deviation.

Model	Acc	Pos err	Ecc err	Params	Flops	Ratio
Dots	1.000	1.11	0.073	14.2K	10.537M	1.00
Dots-ReFT	**0.999**	**3.21**	**0.073**	10.3K	0.332M	0.73
Dots-APoZ	0.997	3.73	0.137	10.2K	0.711M	0.71
Dots-L1-norm	0.791	5.91	0.153	10.3K	0.332M	0.73
Dots-GM	0.745	5.61	0.158	10.3K	0.332M	0.73
Dots-HRank	0.953	4.57	0.156	10.3K	0.332M	0.73
Dots-ReFT-S	0.994	1.25	0.101	10.3K	0.332M	0.73

Table 2: Average accuracy, average dot position error, average eccentricity error, number of parameters and compression ratio with 34 pruned kernels (out of 40) in the first three convolutional layers. Pruning methods have been repeated 20 times. Dot-ReFT-S is the training of Dot-ReFT with random weights. See fig.2 for standard deviation.

that can pre-process frames during acquisition (for example by feeding images to a built-in CNN) may offer a substantial advantage over classical image processing solutions [22, 21]. Usually, such setups require extreme and specialised network pruning approaches, in order to improve both time and memory efficiency. As a case study, we show two practical applications which significantly benefit from the proposed pruning method, especially when implemented on a FPGA device.

The first is realized by a CNN to detect peaks in a one-dimensional light intensity timeserie. This is a typical scenario in 3D reconstruction in which planar laser beams are projected onto the object under study and observed by one or more cameras geometrically calibrated with the laser. The intersection of each laser plane with the object results in a line, usually orthogonal with the pixel arrangement of the linear camera. Therefore, each line produces a spike, or peak, whose position can be easily related with the depth of the object 3D point illuminated by the laser (See for example the signal plotted in the left column of Fig. 1). Our tested model is a relatively simple feed-forward Convolutional

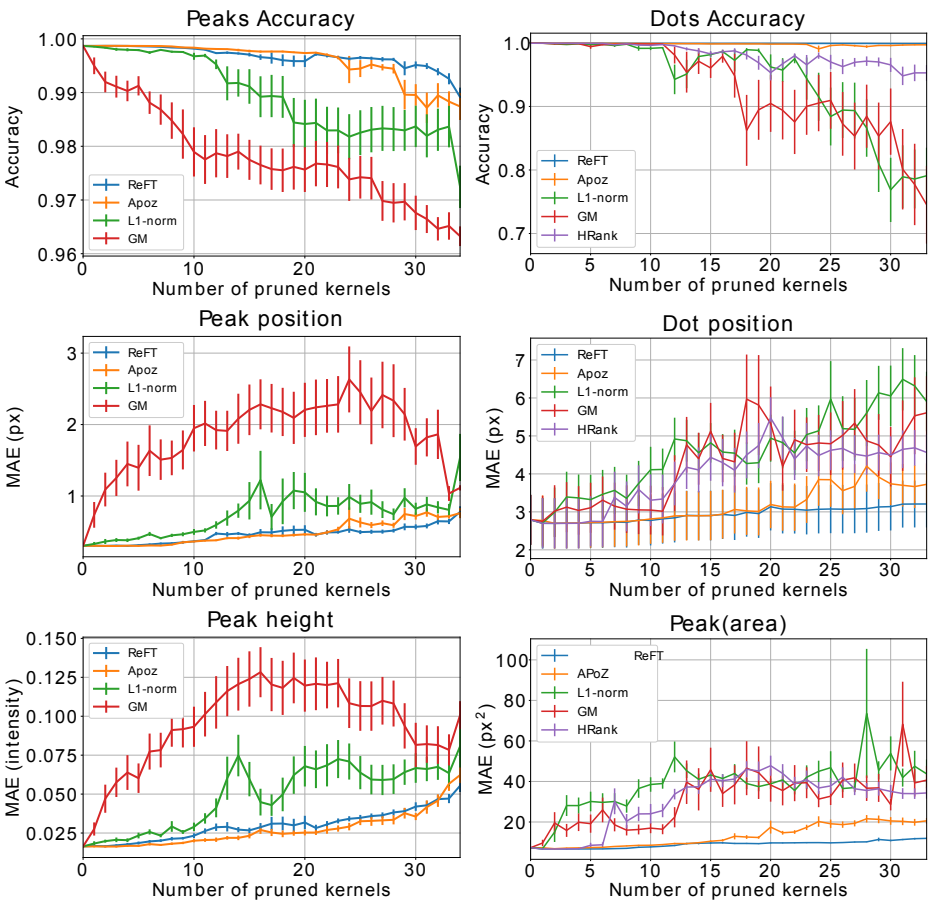


Fig. 2: Comparison of different pruning techniques for peak and dot detection networks (1st and 2nd columns respectively). For each task we report accuracy, MAE of position error and MAE of height and area.

Neural Network taking in input a vector of 1024 intensity values and producing a vector of N output bins, each one containing peak probability, height, location and width of detected peaks. The network is made of 4 convolutional blocks (interleaved by ReLU activations and maxpooling) and then it splits to compute the different losses.

The second case study is the 2-dimensional extension of the linear peak detection network to detect dots in image data (see the right column of Fig. 1 for some examples). The network architecture is essentially equal to the corresponding 1-dimensional case, except for it takes a 2D images as input and 1-dimensional convolutions are replaced with their 2D counterparts.

Both networks provide multiple outputs, namely: the probability of the presence of a peak or dot, the information about peak position, and its height. In the 2D case, we also consider dot area and eccentricity.

Synthetic datasets Our *Peaks* dataset contains 100000 line scan acquisitions divided into training, validation and test set with ratio 80:10:10. Intensity ranges from 0 to 255 in vectors of 1024 values representing a line-camera image. *Dots* dataset was generated by approximating the intensity response with a bivariate Normal function characterized by a certain height (the intensity of the dot), eccentricity (how much the dot deviates from being circular) and angle (the major axis orientation). Signals are assumed to be acquired 80 scanlines at a time and multiple dots can be simultaneously present in each image. It consists of 100000 acquisitions with separated test and validation set of 10000 samples each.

Network training and pruning Both the networks were trained with a learning rate (lr) of $1E-5$, decreased by a factor of 0.1 if the loss does not improve for 3 consecutive epochs. The validation set is used to stop before overfitting. Fine tuning after pruning is performed with $lr = 1E-5$ for a fixed number of epochs $= 3$. We pruned both the *Peaks* and *Dots* networks only from the first 3 blocks, which contain 16, 16 and 8 kernels respectively, leaving at least 1 kernel per block.

Results Figure 2 shows how the performance of the 1D peak detection (top) and 2D dot detection (bottom) networks are affected by the pruning process. Our method lead to a good prediction accuracy even if a severe number of kernels are removed. L1-norm tend to perform worst, with the accuracy dropping significantly especially for the 2D dot detection network. Together with the detection accuracy, position and height are better estimated when the network is pruned with our proposed method. Apoz shows similar performance but higher standard deviation. Table 1 and 2 summarize the performance of the two networks while pruning an optimal number of kernels (33 in this case). It is interesting to notice that both the models pruned with our method (Peak-ReFT and Dots-ReFT) perform better than a model with the same number of kernels trained from scratch (Peak-ReFT-S and Dots-ReFT-S). In other words, it is better to trim a complex network than training a simpler network from the start.

4.2 VGG16 on Cifar10

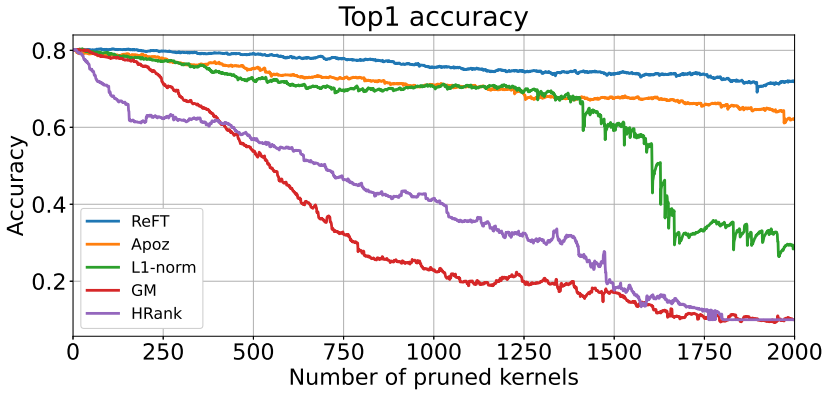
To demonstrate that our method is valid also on more complex CNNs we used VGG16 [25] network on Cifar10 dataset [15]. Cifar10 contains 60000 RGB images belonging to 10 classes and it is often used together with VGG16 to assess the efficiency of compressing methods. VGG16 is a large convolutional network trained on the ILSVR2012 dataset. The network contains 13 convolutional layers with 3 fully connected on top and the activation function for each layer is

a ReLU, except for the final softmax. In order to perform classification on a dataset different from the one the network has been trained on, we need to use *transfer learning*. We exploited the pretrained 13 convolutional layers, to which we attached two more fully connected layers with a small dropout rate and ReLU activations. Finally, we added one last fully connected layer to reduce the output to the correct number of classes. The images are resized from their 32×32 original shape to 48×48, which is the smallest input size required to be able to use all VGG16 convolutional layers. We split the images into 3 separated datasets for train, validation and test with a 60:20:20 ratio. A fast training with learning rate $3E-5$ is performed on Cifar10 dataset, using the validation set to stop the training at a proper time. Every time the pruning algorithm removes a filter we performed a fine tuning step with 5 epochs and a small learning rate set to $1E-6$, followed by another 5 epochs with the same learning rate divided by 10, to recover any loss in accuracy. We tested HRank with the same parameters except for one during the recovery stage, as the pruned kernels are removed in a single pass, and the authors suggest a higher number of epochs (50) to recover the accuracy. The convolutional layers are frozen both at training and pruning time, such that the network can only modify the fully connected weights.

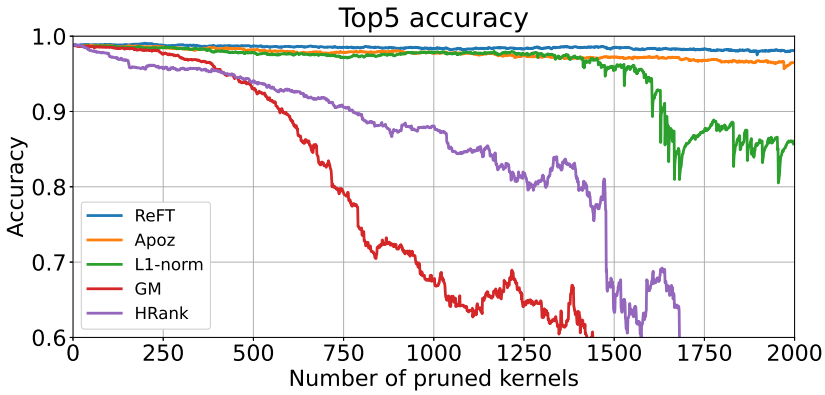
Figure 3 shows the top1 accuracy varying the number of pruned kernels. Our method outperforms the others, with a particularly remarkable difference when more than 300 kernels are removed. Even though ReFT algorithm achieved fair superior performances, some credit has to be given to the other methods. In general, methods that are limited to compute metrics on the filter values (like L1-norm and GM) perform worse than ones analyzing the output of the layers (our method and Apoz). However, the latter are more computational intensive and require a representative dataset to be used, something particularly noticeable when working with images on large networks, such as Cifar10 on VGG16. HRank mitigated this problem by requiring only a small set of images to compute the output responses, but in our tests consistently performed worse than others. HRank, anyway, needs to be run only once, drastically decreasing the computing time during the pruning phase.

Model	Top1	Top5	Params	Flops	Ratio
VGG16	0.801	0.988	15.1M	1.41G	1.000
VGG16-ReFT	**0.780**	**0.986**	11.9M	1.15G	0.789
VGG16-APoZ	0.761	0.983	11.5M	1.23G	0.762
VGG16-L1-norm	0.731	0.979	11.9M	1.15G	0.789
VGG16-GM	0.581	0.944	11.9M	1.15G	0.789

Table 3: Comparison of top1 accuracy, top5 accuracy, number of parameters, FLOPs and compression ratio achieved by filter pruning on VGG16 and Cifar10 after pruning 460 filters.



(a) Top1 accuracy of VGG16 on Cifar10 with different pruning methods.



(b) Top5 accuracy of VGG16 on Cifar10 with different pruning methods.

Fig. 3: Accuracy loss as the number of pruned filter increases for the different methods analysed. The speed in performance degradation is connect to the experimental setup, as the fine-tuning phase does not take into consideration the validation loss for a proper stopping condition, but rather it runs for a fixed number of epochs.

5 Conclusions

We presented a simple CNN pruning method working by ranking the relevance of each convolutional kernel according to the output produced on a representative dataset. Our heuristic is simple to implement but it better preserves the network predicting power compared to similar state-of-the-art approaches while pruning a large amount of kernels.

We analysed two practical case study: peak detection in both 1 and 2 dimensional intensity signals to assess the feasibility on simple networks designed for FPGA hardware. In our tests, we almost halved the number of weights without

a noticeable decrease of prediction accuracy. Even if developed to implement low-level vision tasks, our method has proven to be effective even when applied on classical datasets and network architectures like VGG16 on Cifar10.

References

[1] Blalock, Davis et al. "What is the state of neural network pruning?" In: *arXiv preprint arXiv:2003.03033* (2020).

[2] Cao, S. et al. "ThinNet: An Efficient Convolutional Neural Network for Object Detection". In: *2018 24th International Conference on Pattern Recognition (ICPR)*. 2018, pp. 836–841. DOI: 10.1109/ICPR.2018.8545809.

[3] Courbariaux, Matthieu, Bengio, Yoshua, and David, Jean-Pierre. "Training deep neural networks with low precision multiplications". In: *arXiv preprint arXiv:1412.7024* (2014).

[4] Garg, Isha, Panda, Priyadarshini, and Roy, Kaushik. "A low effort approach to structured CNN design using PCA". In: *IEEE Access* 8 (2019), pp. 1347–1360.

[5] Gasparetto, A. et al. "Cross-Dataset Data Augmentation for Convolutional Neural Networks Training". In: vol. 2018-August. 2018, pp. 910–915. DOI: 10.1109/ICPR.2018.8545812.

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

[7] Han, Song et al. "Learning both weights and connections for efficient neural networks". In: *arXiv preprint arXiv:1506.02626* (2015).

[8] He, Yang et al. "Filter pruning via geometric median for deep convolutional neural networks acceleration". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4340–4349.

[9] Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[10] Howard, Andrew G et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[11] Hu, Hengyuan et al. "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures". In: *arXiv preprint arXiv:1607.03250* (2016).

[12] Iandola, Forrest N et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size". In: (2016).

[13] Jahanshahi, Ali. "TinyCNN: A Tiny Modular CNN Accelerator for Embedded FPGA". In: *arXiv preprint arXiv:1911.06777* (2019).

[14] Jin, Jonghoon, Dundar, Aysegul, and Culurciello, Eugenio. "Flattened convolutional neural networks for feedforward acceleration". In: *arXiv preprint arXiv:1412.5474* (2014).

[15] Krizhevsky, Alex, Nair, Vinod, and Hinton, Geoffrey. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[16] Li, Hao et al. "Pruning filters for efficient convnets". In: *arXiv preprint arXiv:1608.08710* (2016).

[17] Lin, Mingbao et al. "Hrank: Filter pruning using high-rank feature map". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1529–1538.

[18] Lin, Shaohui et al. *Towards optimal structured cnn pruning via generative adversarial learning*. 2019.

[19] Liu, Zhuang et al. "Learning efficient convolutional networks through network slimming". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2736–2744.

[20] Liu, Zhuang et al. "Rethinking the value of network pruning". In: *arXiv preprint arXiv:1810.05270* (2018).

[21] Pistellato, M. et al. "Dynamic optimal path selection for 3D Triangulation with multiple cameras". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9279 (2015), pp. 468–479. DOI: 10.1007/978-3-319-23231-7_42.

[22] Pistellato, M. et al. "Robust joint selection of camera orientations and feature projections over multiple views". In: vol. 0. 2016, pp. 3703–3708. DOI: 10.1109/ICPR.2016.7900210.

[23] Qiu, Jiantao et al. "Going deeper with embedded fpga platform for convolutional neural network". In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2016.

[24] Rastegari, Mohammad et al. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European conference on computer vision*. Springer. 2016, pp. 525–542.

[25] Simonyan, Karen and Zisserman, Andrew. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014).

[26] Suau, Xavier, Apostoloff, Nicholas, et al. "Filter distillation for network compression". In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2020, pp. 3129–3138.

[27] Sze, Vivienne et al. "Efficient processing of deep neural networks: A tutorial and survey". In: *Proceedings of the IEEE* 105.12 (2017).

[28] Zhang, Chaoyun, Patras, Paul, and Haddadi, Hamed. "Deep learning in mobile and wireless networking: A survey". In: *IEEE Communications surveys & tutorials* 21.3 (2019), pp. 2224–2287.

[29] Zhang, Xiangyu et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6848–6856.

[30] Zhu, Michael and Gupta, Suyog. "To prune, or not to prune: exploring the efficacy of pruning for model compression". In: *arXiv preprint arXiv:1710.01878* (2017).