



# Learning Sparse Filters in Deep Convolutional Neural Networks with a $l_1/l_2$ Pseudo-Norm

Anthony Berthelier<sup>1</sup>✉, Yongzhe Yan<sup>1</sup>, Thierry Chateau<sup>1</sup>,  
Christophe Blanc<sup>1</sup>, Stefan Duffner<sup>2</sup>, and Christophe Garcia<sup>2</sup>

<sup>1</sup> Université Clermont Auvergne, Institut Pascal, Clermont-Ferrand, France  
anthony.berthelier@etu.uca.fr

<sup>2</sup> INSA Lyon, LIRIS, Lyon, France

**Abstract.** While deep neural networks (DNNs) have proven to be efficient for numerous tasks, they come at a high memory and computation cost, thus making them impractical on resource-limited devices. However, these networks are known to contain a large number of parameters. Recent research has shown that their structure can be more compact without compromising their performance.

In this paper, we present a sparsity-inducing regularization term based on the ratio  $l_1/l_2$  pseudo-norm defined on the filter coefficients. By defining this pseudo-norm appropriately for the different filter kernels, and removing irrelevant filters, the number of kernels in each layer can be drastically reduced leading to very compact Deep Convolutional Neural Networks (DCNN) structures. Unlike numerous existing methods, our approach does not require an iterative retraining process and, using this regularization term, directly produces a sparse model during the training process. Furthermore, our approach is also much easier and simpler to implement than existing methods. Experimental results on MNIST and CIFAR-10 show that our approach significantly reduces the number of filters of classical models such as *LeNet* and *VGG* while reaching the same or even better accuracy than the baseline models. Moreover, the trade-off between the sparsity and the accuracy is compared to other loss regularization terms based on the  $l_1$  or  $l_2$  norm as well as the SSL [1], NISP [2] and GAL [3] methods and shows that our approach is outperforming them.

**Keywords:** Deep learning · Compression · Neural networks · Architecture

## 1 Introduction

Since the advent of *Deep Neural Networks* (DNNs) and especially *Deep Convolutional Neural Networks* (DCNNs) and their massively parallelized implementations [4, 5], deep learning based methods have achieved state-of-the-art

This work has been sponsored by the Auvergne Regional Council and the European funds of regional development (FEDER).

© Springer Nature Switzerland AG 2021

A. Del Bimbo et al. (Eds.): ICPR 2020 Workshops, LNCS 12661, pp. 662–676, 2021.

[https://doi.org/10.1007/978-3-030-68763-2\\_50](https://doi.org/10.1007/978-3-030-68763-2_50)

performance in numerous visual tasks such as face recognition, semantic segmentation, object classification and detection, etc. [4, 6–9]. Accompanied with the high performance, also high computation capabilities and large memory resources are needed as these models usually contain millions of parameters. These issues prevent them from running on resource-limited devices such as smartphones or embedded devices. Network compression is a common approach in this context, i.e. to reduce the inherent redundancy in the parameters and thus in the computation.

Numerous methods have been developed to obtain compact DNNs. Since a large number of these networks are built upon convolutional layers and since the convolution operations are the most computationally demanding, we are focusing on the reduction of these layers. A simple reduction strategy consists in removing non-relevant filters using pruning methods. For example, Li *et al.* [10] proposed to remove filters that are identified as having a small effect on the output accuracy. Another approach by Luo *et al.* [11] is evaluating information at the filter level using statistical and optimization methods.

Our approach is motivated and inspired by (1) previous works demonstrating the redundancy among the weights of a DCNN [12]; (2) numerous sparsity methods proposed in the literature [13] and (3) the fact that these sparsity methods have rarely been used to remove unimportant weights during training [1]. We therefore propose a new strategy, based on  $l_1/l_2$ -norm, to obtain a subset of kernels with all weights equal to zero (such as that the associated filters can be removed). The main idea is to express the filter reduction problem by introducing sparsity on a set of pseudo-norms computed on each kernel but not directly on the kernels actual values. Figure 1 illustrates the general idea of our method. Each kernel of the network is transformed to a single value using a pseudo-norm. All these values are concatenated into a global vector (its size being the number of filters) called kernel norm vector. Our global kernel-sparsity is defined by the sparsity on this vector and is estimated by a  $l_1/l_2$ -norm ratio. Since a kernel with all weights equal to zero produces a pseudo-norm of zero, the number of filters can be reduced by enforcing sparsity on the kernel norm vector. In this paper, we propose the  $l_1/l_2$ -norm for two reasons: (1) the so-called  $l_1/l_2$ -norm is a simple group norm to implement and (2) the use of the  $l_1$ -norm can increase the performance, interpretability and sparsity of a model [14–16] combined with the  $l_2$ -norm allows to converge to stable solution and maintain sparsity at a good level.

We propose a  $l_1/l_2$ -norm computed on the global vector (vector of kernel pseudo-norms) such that adding this sparsity term to minimize to the global loss will reduce the number of (non-zero) filters of a DCNN. Compared to other approaches, our method presents several advantages:

1. All steps are done during training, i.e. no additional fine-tuning operations are needed.
2. Our method being based on simple  $l_1$  and  $l_2$  norms, is straightforward to implement and compute compared to other methods that remove weights during training.

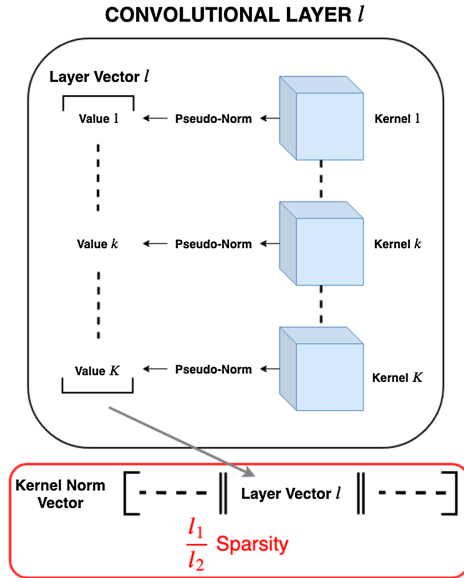


Fig. 1. Visual representation of our method and the computation of the kernel norm vector using a pseudo-norm.

3. As we are keeping track of the evolution of the network at every step during training, it is possible to choose the best model based on a trade-off between compression and accuracy.

In the following, we will first present existing work related to network pruning and weight sparsity, in Sect. 2. In Sect. 3, we describe our  $l_1/l_2$  pseudo-norm method. Finally, in Sect. 4, we show experimental results of our method with *LeNet* and *VGG* network architectures trained on the MNIST and CIFAR-10 datasets. We demonstrate that our method is able to significantly improve the sparsity among convolutional layers in these DCNNs without significant drops in accuracy.

## 2 Related Work

Many studies have been done on DNN compression. Knowledge distillation [17,18] tackles the problem of transferring the encoded information from a bigger model into a smaller one. Lowering numerical precision is also an extensive field [19–21]. Many works, are focused on designing compressed and optimal models architectures. SqueezeNet [22] and MobileNets [23] both propose structures of convolutional layers to improve memory and computation time. Some Neural Architecture Search (NAS) [24–26] methods use reinforcement learning and genetic algorithms to search the best possible networks designs for a given task. Depending on the size of the search space, finding an optimized model with

these methods can be enormously time-consuming. However, the most promising approaches try to reduce the model redundancy and among them: parameter quantization [27, 28] and network pruning [10, 11, 29–31]. Our method can be classified in this last category.

## 2.1 Network Pruning

Pruning methods are aiming to remove unimportant parameters of a neural network. Han *et al.* [27, 29] proposed to prune parameters of *AlexNet* and *VGG* with connection pruning by setting a threshold and removing any parameters under it. As opposed to our method, most of the reduction is done on fully connected layers and not on convolutional layers. However, compression of convolutional layers is essential nowadays as new DNNs are mostly DCNNs with fewer fully connected layers *e.g.*, only 3.99% parameters of *Resnet* [9]. Closer to our approach, structured pruning methods are removing directly structured parts *e.g.*, kernels or layers, to compress CNNs. Li *et al.* [10] used  $l_1$ -norm to remove filters. He *et al.* [32] used a LASSO regression based channel selection to prune filters. Channel pruning methods are preferred on widely-used DCNNs. For example, the selection of unimportant feature maps can be done using  $l_1$ -regularization [33].

These past few years, numerous networks compression algorithms using pruning methods and achieving state-of-the-art results have emerged. Yu *et al.* [2] proposed a neurons importance score propagation (NISP) method based on the response of the final layers to evaluate the pruning impact of the prior layers. Zhuang *et al.* [34] developed discrimination-aware losses in order to determine the most useful channels in intermediate layers. Some methods such as Filter Pruning Via Geometric Median (FPGM) [35] are not focused on pruning filters with less importance but only by evaluating their redundancy. Similarly, Lin *et al.* [3] tackled the problem of redundant structures by proposing a generative adversarial learning method (GAL) (not only to remove filters, but also branches and blocks).

Still, standard pruning methods usually construct non structured and irregular connectivity in a network, leading to irregular memory access. In most of these approaches, the DNN is trained first. Then each parameter is evaluated to understand if it brings information to the network. If not, the parameter is removed. Therefore, a fine-tuning needs to be performed afterwards to restore the model accuracy. These steps take time. Most of them are done offline and need costly reiterations of decomposing and fine-tuning to find an optimal weight approximation maintaining high accuracy and high compression rate. Unlike these methods, our approach is able to directly increase the sparsity of the network during training, identifying which kernels to prune without any considerable extra computational overhead.

## 2.2 Weight Sparsity

An important factor for the compression of a model is its sparsity *i.e.* the number of parameters set to zero. However, this sparsity must be structured in order to be memory-efficient and time-efficient. Liu *et al.* [36] obtained a sparsity of 90% on *AlexNet* with only 2% accuracy loss using sparse decomposition and a sparse matrix multiplication algorithm. This method also employed group Lasso [37], an efficient regularization to learn sparse structures. It is also used by Wen *et al.* [1] to regularize the structure of a DNN at different levels (*i.e.* filters, channels, filter shapes and layer depth). This approach leads to DNNs with reduced computational cost and efficient acceleration due to the structured sparsity induced by the method. We propose to use a different type of regularization based on the norm ratio  $l_1/l_q$  [13,38]. It allows to dynamically maximize the sparsity of a model with one hyper-parameter ( $q$ ) without additional iterations and severe drops in accuracy while being straightforward to implement.

## 3 Training with Kernel-Sparsity

We mainly focus on inducing sparsity on convolutional layers to regularize and compress the structure of DCNNs during the training steps. We propose a generic method to regularize DCNNs using the  $l_1/l_2$  pseudo-norm.

### 3.1 Kernel-Sparsity Regularization

Let  $\mathcal{N}$  be a DCNN with  $L$  convolutional layers. We define  $W^{l,k}$  as the  $k^{th} \in \{1, ..N_k^l\}$  3d-tensor (kernel) associated with the  $l^{th}$  convolutional layer. Thus, a weight of kernel  $k$  in the convolutional layer  $l$  is defined as: and  $W_{w,h,c}^{l,k} \in \mathbb{R}^{N_w^l, N_h^l, N_c^l}$  the (width, height, channel) weight of kernel  $k$  of layer  $l$ .

$$W_{w,h,c}^{l,k} \in \mathbb{R}^{N_w^l, N_h^l, N_c^l} \tag{1}$$

Here,  $w \in \{1, ..N_w^l\}$  is the column,  $h \in \{1, ..N_h^l\}$  is the row and  $c \in \{1, ..N_c^l\}$  is the channel index of the  $k^{th}$  kernel matrix in the convolutional layer  $l$ . The key idea is to express sparsity on pseudo-norms of kernels. Let  $n_k^l$  be the pseudo-norm defined by the  $l_1$ -norm of the flattened kernel  $W^{l,k}$ :

$$n_k^l \doteq \sum_{w=1}^{N_w^l} \sum_{h=1}^{N_h^l} \sum_{c=1}^{N_c^l} \frac{|W_{w,h,c}^{l,k}|}{N_k^l} \tag{2}$$

The vector  $\vec{N}^l$  concatenates, for layer  $l$ , the  $N_k^l$  norms  $n_k^l$ :

$$\vec{N}^l \doteq \left\| \begin{matrix} N_k^l \\ n_k^l \end{matrix} \right\|_{k=1} \tag{3}$$

We introduce kernel-sparsity for a layer as a value linked to the number of kernels of this layer with all weights equal to zeros. Therefore, the kernel-sparsity of layer  $l$  can be linked with the number of values of the vector  $\vec{N}^l$  equal to zero. Global kernel-sparsity can be expressed from the concatenation of vectors  $\vec{N}^l$  for each layer:

$$\vec{N} \doteq \left\| \left\| \vec{N}^l \right\|_{l=1}^L \right\| \tag{4}$$

For better understanding, we visualize these operations in Fig. 2. In order to normalize the value of N, each of its component is divided by the number of values (or norms) that it contains. Finally, the global kernel-sparsity is defined by the sparsity of  $\vec{N}$  and can be estimated by a  $l_1/l_2$  ratio function:

$$\mathcal{L}_s \doteq \frac{\vec{N}_1}{\vec{N}_2} \tag{5}$$

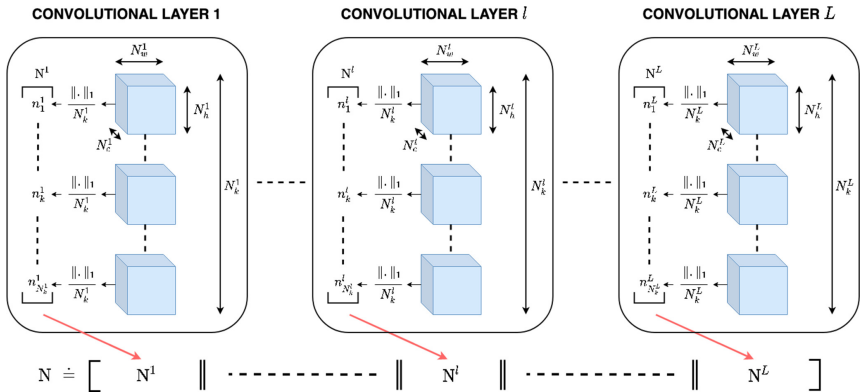
Minimizing this term will encourage zero-valued coefficients (numerator), corresponding to the different kernels, while keeping the remaining coefficients at large values (denominator), thus producing convolution layers with few non-zero kernels.

### 3.2 Training with Kernel-Sparsity Regularization

Let  $\mathcal{L}_{\mathcal{N}}$  be the loss function that is minimized to find the optimal weight configuration for a given task (e.g. cross entropy). We propose to simply add the kernel-sparsity regularization term weighted by the coefficient  $\lambda \in \mathbb{R}$ :

$$\mathcal{L}_{all} = \mathcal{L}_{\mathcal{N}} + \lambda \mathcal{L}_s . \tag{6}$$

We will discuss how to set an appropriate values of  $\lambda$  in the experimental section.



**Fig. 2.** Visualization of the computation of the kernels pseudo-norm and how the global kernel norm vector  $\vec{N}$  is obtained.

### 3.3 Setting Kernels to Zero

Our method induces sparsity in a DCNN, i.e. the pseudo-norm regularization pushes some kernels to have only zero-valued coefficients. However, in practice, during optimization, the actual values of these kernels will not be exactly zero but very small. Thus, to compress the network effectively, our approach identifies these kernels during training and forces them to be zero in order to remove them.





More specifically, the algorithm works as follows: each pseudo-norm of the kernels is contained in the global kernel pseudo-norm vector  $\vec{N}$ . Thus, at each epoch, we normalize the values of  $\vec{N}$  so that  $\sum_{i=1}^K \vec{N}_i = 1$ . Sorting these vectors in ascending order will allow us to objectively determine which pseudo-norms are the smallest. We then define a percentage (or a threshold) under which the cumulative sum of these sorted values is judged too small to be kept, i.e. the corresponding filters are considered unimportant and set to zero. Once the weights of a kernel are set to zero, they are keeping this value until the end of the training, and these parameters are no longer updated. This ensures that the potential errors and imprecision introduced by removing these kernels can be compensated by the remaining kernels during the training converging to a stable solution with high accuracy.

To summarize, our approach consists of two steps at each epoch:

1. The  $l_1/l_2$  pseudo-norm is computed on each kernel of the model and is integrated to the loss function. Thus the training stage is minimizing the loss function and inducing sparsity at the kernel level, pushing some weights to have a near zero value.
2. Sort kernels according to their ascending normalized pseudo-norm and compute a cumulative sum vector from the sorted normalized pseudo-norm vector. The kernels participating to the cumulative sum under a threshold  $t$  are removed. This set of operations aims at keeping kernels that produce more than  $t\%$  of the global norm.

## 4 Experiments

We evaluated the performance of the  $l_1/l_2$  pseudo-norm on two classification models (*LeNet* and *VGG*) and two datasets: MNIST and CIFAR-10. Our method is implemented in *Pytorch*, running on various Nvidia GPUs using CUDA. The weights of the networks are initialized randomly and hyper-parameters are selected manually for optimal results. The chosen  $\lambda$  value is the one allowing the model to have an accuracy close to its baseline accuracy while sparsifying the most the kernels norms. In all the experiments, the threshold under which the kernels are removed by evaluating the cumulative sum of the smallest norms is set to 1%. We found that this value was the best trade-off between a converging accuracy of the models and a slow removal of the kernels during the training phase.

| EPOCH | KERNELS   | $\mathcal{L}_s$ |
|-------|---|-----------------|
| 0     |  | 0.0226          |
| 4     |  | 0.0220          |
| 20    |  | 0.0189          |
| 130   |  | 0.0118          |

**Fig. 3.** Evolution of the 20 kernels of the first convolutional layer of *LeNet* and the global kernel-sparsity regularization term  $\mathcal{L}_s$  during a training on the MNIST dataset. For a better visualization, each kernel is flattened from 3 dimensions to 2 dimensions.

### 4.1 Experiments on LeNet

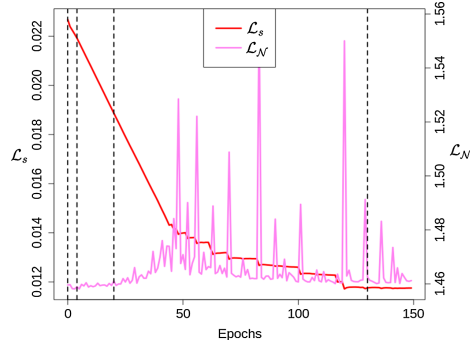
In the experiments with *LeNet* [39], we investigate the effectiveness of the  $l_1/l_2$  pseudo-norm on the MNIST and CIFAR-10 datasets. In order to compare our results with state-of-the-art methods such as SSL [1], NISP [2] and GAL [3], we decide to chose the *LeNet* model implemented by *Caffe*. All these methods are using evaluation and regression at different level i.e Lasso-group regression at different level of a convolutional layer in the SSL method, which makes it more complex to implement than our method. There is no data augmentation for the training on both datasets.

***LeNet on MNIST:*** As previously described, the  $l_1/l_2$  pseudo-norm is applied on the filters of a DCNN to penalize them. Hence our method is inducing sparsity among the filters of the convolutional layers in *LeNet*. To visualize the effect on our approach on the kernels, Fig. 3 shows the evolution of the kernels of the first convolutional layer of *LeNet* during a training on MNIST with our kernel-sparsity regularization. We see that the kernel-sparsity term  $\mathcal{L}_s$  is decreasing epoch after epoch and that the number of filters in the layer is also decreasing with it. The complete evolution of the kernel-sparsity  $\mathcal{L}_s$  can be seen on Fig. 4.  $\mathcal{L}_s$  being computed on the pseudo-norm of the kernels and kernels weights being set to zero over time: this result shows the effectiveness of our method.

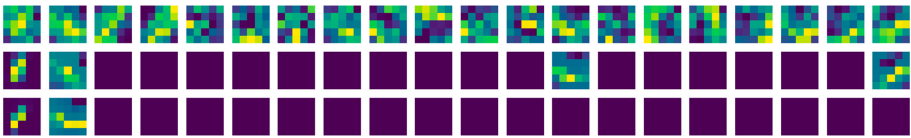
Table 1 summarizes the results on MNIST of different methods. In the best case scenario that we have tested, the  $l_1/l_2$  pseudo-norm with a  $\lambda$  value set to 0.5 is able to achieve an accuracy better than the baseline by 0.2%. Furthermore the number of filters is dropping drastically in both convolutional layers respectively from 20 to 5 and from 50 to 18. Compared to the other state-of-the-art methods, the  $l_1/l_2$  pseudo-norm is able to achieve a better accuracy while penalizing more filters too. We also compare our method to the  $l_1$ -norm and  $l_2$ -norm. During our evaluations, both of these norms were able to reach a higher level of sparsity by setting to zero more kernels. However they were never able to reach the same or a better level of accuracy than the baseline.

To visualize the effect of our method on the parameters, we show the learned filters of the first convolutional layer in Fig. 5. For  $\lambda = 0.5$  and for different level of sparsity, it can be seen that the number of remaining filters can be set to only 2 or 4. Furthermore between the baseline and our method, the accuracy is the same





**Fig. 4.** Evolution of the cross-entropy loss function  $\mathcal{L}_N$  and the global kernel-sparsity regularization term  $\mathcal{L}_s$  during training with  $\lambda = 0.5$ . Evaluations are done on *LeNet* on the MNIST dataset. Vertical lines show which epochs were taken to construct Fig. 3.



**Fig. 5.** Learned filters of the first convolutional of *LeNet* on MNIST. Top is *LeNet* baseline, middle and bottom are  $l_1/l_2$ -norm with  $\lambda = 0.5$  and different level of sparsity.

or is increased. This shows that there is effectively a large amount of redundancy between filters and that most of them are not required. Moreover, compared to the baseline, it seems that the remaining filters are more structured, with more regular patterns. This assumption seems especially true when only two filters are remaining. Thus, we arrive at the same conclusion than [1]: the baseline has a high freedom in the parameter space and our method is able to obtain the same accuracy by optimizing the filters into more regularized patterns.

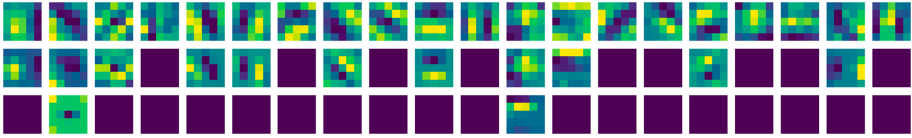
***LeNet on CIFAR-10:*** In order to test the  $l_1/l_2$  pseudo-norm and visualize its effect on a more difficult classification task than MNIST, we decided to use the CIFAR-10 dataset with the same *LeNet* model. The results are summarized in Table 2. The baseline *LeNet* is not performing as well on CIFAR-10 than it is performing on MNIST, i.e. the classification accuracy is only around 70%. As a result, the accuracy of our approach also drops but the  $l_1/l_2$  pseudo-norm is still able to perform well on this model, even for this classification task. With a  $\lambda$  value set to 0.7, we are able to decrease the number of filters in the first and the second convolutional layers respectively from 20 to 10 and from 50 to 25, which means that half of the filters of *LeNet* are removed. With this configuration, our method performs 1.7% worse than the baseline. We were able to remove up to 80% of the filters in our experiments, but the resulting accuracy was too low to be interesting (more than 20% behind the baseline). Hence, more filters

**Table 1.** Results after penalizing unimportant filters in *LeNet* on MNIST. Baseline is the simple *LeNet* Caffe model.  $l_1$  and  $l_2$  are the best results found by using the  $l_1$ -norm and  $l_2$ -norm regularization on the kernels. SSL, NISP and GAL are the pruning methods respectively from [1–3].  $l_1/l_2$  is our method with  $\lambda = 0.5$ .

| Method    | $\lambda$ | Error       | Conv1 filter #<br>(Sparsity) | Conv2 filter #<br>(Sparsity) | Total sparsity |
|-----------|-----------|-------------|------------------------------|------------------------------|----------------|
| Baseline  | –         | <b>0.9%</b> | 20                           | 50                           | <b>0%</b>      |
| $l_1$     | 0.5       | 1.2%        | 4<br>(80%)                   | 5<br>(90%)                   | 87.1%          |
| $l_2$     | 0.5       | 1.2%        | 3<br>(85%)                   | 5<br>(90%)                   | 88.6%          |
| SSL 1     | –         | <b>0.8%</b> | 5<br>(75%)                   | 19<br>(62%)                  | <b>65.7%</b>   |
| SSL 2     | –         | 1.0%        | 3<br>(85%)                   | 12<br>(76%)                  | 78.6%          |
| NISP      | –         | <b>0.8%</b> | 10<br>(50%)                  | 25<br>(50%)                  | <b>50.0%</b>   |
| GAL       | –         | 1.0%        | 2<br>(90%)                   | 15<br>(70%)                  | 75.7%          |
| $l_1/l_2$ | 0.5       | <b>0.7%</b> | 5<br>(75%)                   | 18<br>(64%)                  | <b>67.1%</b>   |

are needed in order to classify correctly the CIFAR-10 dataset compared to the MNIST dataset. The best trade-off between filters and accuracy that we found was still with a value of  $\lambda$  set to 0.7. In both convolutional layers, the number of filters is dropping respectively from 20 to 14 and from 50 to 30. This means that our method is able to zero out more than a third of the filters with only a drop of 0.9% in accuracy. Compared to the  $l_1$ -norm and the  $l_2$ -norm, the  $l_1/l_2$  pseudo-norm also shows good results. Indeed, both the  $l_1$ -norm and  $l_2$ -norm were unable to reach the same level of accuracy and setting to zero as many filters as the  $l_1/l_2$  pseudo-norm can do.

As previously done with the MNIST dataset, we visualize the learned filters of the first convolutional layer in Fig. 6. From this visualization, we can draw the same conclusion than with the MNIST dataset. The more we are removing filters, the more the remaining ones seem to have a defined structure, as opposed to the baseline where each of the filters seems blurry. It is even more remarkable when we let our algorithm run until only a couple of filters are remaining. Even if the model does not reach a satisfactory accuracy, the two remaining filters have learned remarkable patterns. Thus, the  $l_1/l_2$  pseudo-norm is still able to smooth a high freedom of parameter space into fewer filters with more regularized patterns.



**Fig. 6.** Learned filters of the first convolutional layer of *LeNet* on CIFAR-10. Top is *LeNet* baseline, middle and bottom are  $l_1/l_2$ -norm with  $\lambda = 0.7$  and different level of sparsity.

**Table 2.** Results after penalizing unimportant filters in *LeNet* on CIFAR-10. Baseline is the simple *LeNet* Caffe model.  $l_1/l_2$  is our method with different coefficient of regulation  $\lambda = 0.7$ .

| Method    | $\lambda$ | Error        | Conv1 filter #<br>(Sparsity) | Conv2 filter #<br>(Sparsity) | Total sparsity |
|-----------|-----------|--------------|------------------------------|------------------------------|----------------|
| Baseline  | -         | <b>28.4%</b> | 20                           | 50                           | <b>0%</b>      |
| $l_1$     | 0.7       | <b>35.4%</b> | 7<br>(65%)                   | 14<br>(72%)                  | <b>70.0%</b>   |
| $l_2$     | 0.7       | 29.8%        | 12<br>(40%)                  | 24<br>(52%)                  | 48.6%          |
| $l_1/l_2$ | 0.7       | 30.1%        | 10<br>(50%)                  | 25<br>(50%)                  | 50.0%          |
| $l_1/l_2$ | 0.7       | <b>29.3%</b> | 14<br>(30%)                  | 30<br>(40%)                  | <b>37.1%</b>   |

### 4.2 VGG on CIFAR10

To demonstrate the generalization of our method on larger DNNs, we evaluate the performance of our method on the well-known *VGG* [6], a deeper model than *LeNet*, with several convolutional layers. A *VGG* model can have different sizes, notably depending on the number of layers. We chose the *VGG11* model with a total of 8 convolutional layers. We implemented it using *Pytorch*, running on various Nvidia GPUs using CUDA. The model is trained without data augmentation and evaluated on the CIFAR-10 dataset. In this experiment, the kernels pseudo-norms are not normalized on the full network, which explains why the  $\lambda$  values are smaller than the ones used with *LeNet*.

With *LeNet*, the  $l_1/l_2$  pseudo-norm method was applied on only 2 convolutional layers, with 50 filters at most in the second convolutional layer. In *VGG11* our method is applied on 8 different convolutional layers with a number of filters set to 64 in the first convolutional layer and a maximum of 512 filters in the last four convolutional layers. The results are shown in Table 3. The baseline model, with all the filters and a classical loss function (cross-entropy), obtains an error of 17.6% on the test dataset. Using the  $l_1/l_2$  pseudo-norm with a  $\lambda$  set to 0.005, the model achieves a classification accuracy roughly 1% inferior to the baseline. However the number of filters is vastly reduced. Moreover, it seems that the deeper we go in the network, the more the proportion of filter sets to

zero is important. For example, the second convolutional layer has around 10% of its filters set to zero while the last convolutional layer has over 65% of filters set to zero. Thus we could deduce that the last convolutional layers keep less important information for the model than the first ones or that there is more redundancy in the last layers. However, the first convolutional layer seems to be an exception as approximately half of its filters can be removed. We suppose that the shapes learned in the first layer are not decisive for the model and can be balanced by the following layers and the more defined shapes that they have assimilated.

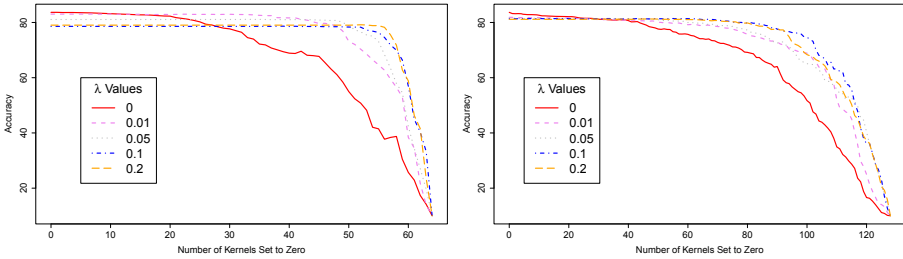
By decreasing the  $\lambda$  coefficient to 0.001, we confirm the results that the last convolutional layers seem to contain more filters with non decisive or redundant information than the first ones. Indeed, only the last two layers have filters set to zero. But more importantly, the removal of a few filters in the last two convolutional layers leads to a classification error of only 16.8%, which is 0.8% less than the baseline. Thus, our method, by only removing a few filters, is able to achieve a better accuracy than the baseline model. Compared to the  $l_1$ -norm and the  $l_2$ -norm, the  $l_1/l_2$  pseudo-norm is also performing well. The  $l_1$ -norm is able to zero out numerous filters but is unable to achieve a correct level of accuracy, always performing worse than the baseline or our approach. Nearly the same conclusions can be drawn from the  $l_2$ -norm. Under certain conditions, the  $l_2$ -norm is able to zero out slightly more filters than our method in the last convolution layers. However, the models are not able to obtain a satisfying accuracy, always around 1% behind the baseline.

In order to conclude this study, we visualize in Fig. 7 the evolution of the accuracy of the model against the number of kernel set to zero in the first and second convolutional layers for different coefficient of regularization  $\lambda$ . When  $\lambda = 0$ , the  $l_1/l_2$  pseudo-norm is not taken into account, resulting into the baseline model. The order that the filters are set to zero is determined by the filters

**Table 3.** Results after penalizing unimportant filters in *VGG11* on CIFAR-10. Baseline is the *VGG11* network baseline.  $l_1/l_2$  is our method with different coefficient of regulation  $\lambda$ .

| Method    | $\lambda$ | Error        | Conv1 to conv 8 filter<br>#                     | Total sparsity |
|-----------|-----------|--------------|---|----------------|
| Baseline  | -         | <b>17.6%</b> | 64 - 128 - 256 - 256 -<br>512 - 512 - 512 - 512 | <b>0%</b>      |
| $l_1$     | 0.0001    | 19.3%        | 52 - 128 - 255 - 256 -<br>175 - 147 - 97 - 123  | 55.2%          |
| $l_2$     | 0.005     | <b>18.2%</b> | 64 - 128 - 256 - 256 -<br>511 - 474 - 434 - 299 | <b>12%</b>     |
| $l_1/l_2$ | 0.005     | 18.8%        | 35 - 115 - 238 - 176 -<br>354 - 195 - 190 - 175 | 46.3%          |
| $l_1/l_2$ | 0.001     | <b>16.8%</b> | 64 - 128 - 256 - 256 -<br>512 - 512 - 510 - 380 | <b>5%</b>      |

pseudo-norm arranged by ascending order. These tests are done at a single convolutional layer level. Meaning that during training, the only filters that are evaluated and set to zero are the ones belonging to the studied layer. The other layers are remaining untouched. We visualize that for both layers, we are able to set numerous filters to zero without a noticeable decrease of the accuracy, even when our method is not active. This result shows that there is unimportant information in the layer and that it is possible to remove it, even if there are no methods that are defined to emphasize this phenomenon. With the implementation of the  $l_1/l_2$  pseudo-norm ( $\lambda > 0$ ), we see that (1) more kernels are set to zero before the beginning of the accuracy drop compared to the baseline and (2) a greater  $\lambda$  value means that more kernels are zeroed-out but at the price of an inferior accuracy. Based on these conclusions, our method is increasing the sparsity of the filters within a layer, shifting information between them in order to centralize the information. However this sparsity has its limits. The more we force it (with a significant  $\lambda$  value), the more we increase the chances to lose important information that could be never recovered.



(a) First convolutional layer of *VGG11*      (b) Second convolutional layer of *VGG11*

**Fig. 7.** Visualization of the effect of setting kernels to zero in the first two convolutional layers of *VGG11* against the accuracy of the network. Each line represents a different value for the coefficient of regulation  $\lambda$  of the  $l_1/l_2$  pseudo-norm method.

## 5 Conclusion

In this work, we proposed a new regularization approach for inducing kernel sparsity in DCNNs based on the  $l_1/l_2$  pseudo-norm. This method reorganizes the weights of the convolutional layers in order to learn more compact structures during training. These compact DCNNs can reach almost the same accuracy as the original models and in some cases even perform better. Our experiments have demonstrated the benefits of our approach and its generalization to deep structures: it is straightforward to implement, it operates during the training process and it is possible to choose between the compactness and the accuracy of the model. So far, we have only applied our method to classification problems. To go beyond, in the future, the method needs to be applied on deeper models such as Resnet [9] and bigger datasets. Autoencoders, fully convolutional networks and segmentation problems are also an important focus. Furthermore, it

would be interesting to generalize the  $l_1/l_2$ -norm to the  $l_1/l_q$ -norm to study its properties in more detail and improve on different model structures and learning problems.

## References

1. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. arXiv preprint [arXiv:1608.03665](https://arxiv.org/abs/1608.03665) (2016)
2. Yu, R., et al.: NISP: pruning networks using neuron importance score propagation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9194–9203 (2018)
3. Lin, S., et al.: Towards optimal structured CNN pruning via generative adversarial learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2790–2799 (2019)
4. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **25**, 1097–1105 (2012)
5. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
6. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale Image recognition. arXiv preprint [arXiv:1409.1556](https://arxiv.org/abs/1409.1556) (2015)
7. Szegedy, C., et al.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
8. He, K., Sun, J.: Convolutional neural networks at constrained time cost. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5353–5360 (2015)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
10. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient ConvNets. arXiv preprint [arXiv:1608.08710](https://arxiv.org/abs/1608.08710) (2017)
11. Luo, J. H., Wu, J., Lin, W.: Thinet: a filter level pruning method for deep neural network compression. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 5058–5066 (2017)
12. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. arXiv preprint [arXiv:1405.3866](https://arxiv.org/abs/1405.3866) (2014)
13. Bach, F., Jenatton, R., Mairal, J., Obozinski, G.: Optimization with sparsity-inducing penalties. arXiv preprint [arXiv:1108.0775](https://arxiv.org/abs/1108.0775) (2012)
14. Huang, J., Zhang, T.: The benefit of group sparsity. *Ann. Statist.* **38**(4), 1978–2004 (2010)
15. Turlach, B., Venables, W., Wright, S.: Simultaneous variable selection. *Technometrics* **47**(3), 349–363 (2000)
16. Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. *J. Roy. Stat. Soc. B (Stat. Methodol.)* **68**(1), 49–67 (2006)
17. Dauphin, Y.N., Bengio, Y.: Big neural networks waste capacity. arXiv preprint [arXiv:1301.3583](https://arxiv.org/abs/1301.3583) (2013)
18. Ba, L.J., Caruana, R.: Do deep nets really need to be deep? arXiv preprint [arXiv:1312.6184](https://arxiv.org/abs/1312.6184) (2014)
19. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International Conference on Machine Learning, pp. 1737–1746 (2015)

20. Courbariaux, M., Bengio, Y., David, J.P.: Training deep neural networks with low precision multiplications. arXiv preprint [arXiv:1412.7024](https://arxiv.org/abs/1412.7024) (2014)
21. Williamson, D.: Dynamically scaled fixed point arithmetic. In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings, pp. 315–318 (1991)
22. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size. arXiv preprint [arXiv:1602.07360](https://arxiv.org/abs/1602.07360) (2016)
23. Howard, A.G., et al.: Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv preprint [arXiv:1704.04861](https://arxiv.org/abs/1704.04861) (2017)
24. Miikkulainen, R., et al.: Evolving deep neural networks. In: Artificial Intelligence in the Age of Neural Networks and Brain Computing, pp. 293–312 (2017)
25. Tan, M., Chen, B., Pang, R., Vasudevan, V., Le, Q.V.: MnasNet: platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2820–2828 (2019)
26. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: AMC: autoML for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 784–800 (2018)
27. Han, S., Mao, H., Dally, W. J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint [arXiv:1510.00149](https://arxiv.org/abs/1510.00149) (2016)
28. Choi, Y., El-Khamy, M., Lee, J.: Towards the limit of network quantization. arXiv preprint [arXiv:1612.01543](https://arxiv.org/abs/1612.01543) (2017)
29. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural network. arXiv preprint [arXiv:1506.02626](https://arxiv.org/abs/1506.02626) (2015)
30. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM J. Emerg. Technol. Comput. Syst. **13**(3), 1–18 (2017)
31. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. arXiv preprint [arXiv:1611.06440](https://arxiv.org/abs/1611.06440), 3 (2017)
32. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1389–1397 (2017)
33. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2736–2744 (2017)
34. Zhuang, Z., et al.: Discrimination-aware channel pruning for deep neural networks. arXiv preprint [arXiv:1810.11809](https://arxiv.org/abs/1810.11809) (2018)
35. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4340–4349 (2019)
36. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 806–814 (2015)
37. Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. J. Roy. Stat. Soc. B (Stat. Methodol. **68**(1), 49–67 (2006)
38. Liu, J., Ye, J.: Efficient l1/lq norm regularization. arXiv preprint [arXiv:1009.4766](https://arxiv.org/abs/1009.4766) (2010)
39. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)