A Multilayer Network Based Approach to Represent, Explore and Manipulate Convolutional Neural Networks

Alessia Amelio[1], Gianluca Bonifazi[2], Enrico Corradini[2], Domenico Ursino[2*], and Luca Virgili[2]

[1] DIG, University "G. D'Annunzio" of Chieti-Pescara
[2] DII, Polytechnic University of Marche
[*] Contact Author
a.amelio@unich.it; g.bonifazi@univpm.it; e.corradini@pm.univpm.it; d.ursino@univpm.it; l.virgili@pm.univpm.it

Abstract

Deep learning techniques and tools have experienced an enormous growth and a widespread diffusion in recent years. At the same time, the need for tools able to explore, understand and possibly manipulate the internal structure of a deep learning model has strongly emerged. In this paper, we provide a contribution in that direction. In particular, we propose an approach to map a deep learning model into a multilayer network. Our approach is thought for a very common family of deep learning networks, namely Convolutional Neural Networks (CNNs), but can be easily extended to other families. As a second main contribution, in order to show how our approach enables the exploration and manipulation of deep learning networks, we illustrate an approach for compressing a CNN. It detects whether there are convolutional layers that can be pruned without losing too much information and, if so, returns a new CNN obtained from the original one by pruning such layers. This paper is completed with an experimental campaign aimed at verifying the appropriateness and effectiveness of the proposed approaches.

Keywords: Deep Learning; Convolutional Neural Networks; Complex Networks; Multilayer Networks; Mapping CNNs into Multilayer Networks; Convolutional Layer Pruning

1 Introduction

In recent years, deep learning models have been introduced in different application areas for their ability to solve different kinds of optimization problem, such as document and text processing, object recognition in images and videos, image generation, speech and language recognition and translation [15]. Due to the increasing complexity of these tasks, more demanding models have been required in order to achieve the best performances, such as residual networks [21], inception networks [59] and dense networks [28]. For these models, it is important to use more computational power with Graphical Processing Units (GPU) for speeding up the time-consuming training process. Furthermore, in real-time decision making, a deep model with many parameters requires more time and resources

to process its input, with further requirements in terms of energy and space. Finally, in mobile and edge devices, deep learning is a big opportunity but, at the same time, a big issue because, in this scenario, the computational power, storage capacity and energy are limited [49, 13].

Several authors have begun to highlight the importance of reducing the size of deep networks and accelerating the models in terms of network structure and knowledge [11, 12, 45]. Accordingly, most effort has been performed to introduce efficient and reduced-in-size ad-hoc deep network architectures, such as Mobile networks [27], SqueezeNet [30], ShuffleNet [68] and ESPNet [48]. Furthermore, different methods for reducing the model size whilst preserving the loss of performance have been proposed. The latter refers to four main categories of methods, i.e., *(i)* pruning, *(ii)* quantization, *(iii)* low-rank factorization, and *(iv)* knowledge distillation [13, 60].

To maximize the effectiveness of these methods, it is extremely important to be able to explore the various layers and components of a deep learning architecture. Such an exploration should allow for the identification of the most important components, the detection of interesting patterns and features, the tracking of information flow, the understanding of which parts of the network can be preserved, which can be replaced or removed, and so forth.

In this paper, we want to make a contribution in this setting. In particular, we believe that complex networks, and specifically multilayer networks, can be a very useful tool to represent, analyze, explore and manipulate deep learning networks. Based on this intuition, we first propose an approach to map deep learning networks into complex networks and then we use the latter to explore and manipulate the former. More specifically, we focus on one family of deep learning networks, namely Convolutional Neural Networks (hereafter, CNNs) [34], although the proposed approach could be extended to other ones. The complex network we use is a multilayer network [37]; it is sufficiently articulated to capture all aspects characterizing CNNs. Our mapping approach, which is the first main contribution of this paper, aims to map every aspect of a CNN (nodes, layers, filters, weights, etc.) in the four main components of a multilayer network, namely nodes, arcs, arc weights and layers.

Once we have applied our mapping approach and obtained a multilayer network, the latter can be used for various, both exploratory and manipulative, purposes. To give an idea of its potential, we will use it as a support structure for a convolutional layer pruning approach [9]. The objective of this approach is identifying whether there are layers of a CNN that can be pruned without losing too much information and, in the affirmative case, returns a new CNN obtained from the original one by pruning these layers. Such a pruning approach represents the second main contribution of this paper.

The outline of this paper is as follows: in Section 2, we present the Related Literature. In Section 3, we illustrate our approach for mapping a CNN into a multilayer network. In Section 4, we describe our CNN layer pruning approach based on the multilayer network. In Section 5, we present the experiments we performed to evaluate our approach. Finally, in Section 6, we draw our conclusion and indicate some possible future developments of our research efforts.

2 Related Literature

In the last years, different approaches have been introduced in the literature for the pruning, the quantization, the low-rank factorization of neural networks, as well as for extracting knowledge from them. Pruning methods can be classified as: *(i) weight pruning*, where redundant connections, or

connections having a weight below a threshold, are pruned; *(ii) neuron pruning*, where redundant neurons, together with incoming and outgoing connections, are pruned; *(iii) filter pruning*, where the least relevant layers, according to a given ranking, are pruned; *(iv) layer pruning*, where pruning of some layers is performed [13, 64]. In this research area, the approach of [58] prunes network connections according to their impact on the training error. Specifically, it removes the connections units with the least impact on the error; after this, it adopts the back-propagation algorithm for re-training the network. The approach of [56] also removes redundant neurons by pruning the weights providing the minimum change in the output activation of neurons. Sparsely connected networks are introduced in [6]. The approach described in this paper randomly deletes connections from a dense layer using a new sparse weight matrix. Furthermore, it proposes an efficient hardware architecture for reducing memory usage. In [7], the authors propose an approach for an efficient pruning of parameters based on the correlation between neuron activations in the inner layer and the increase of neuron correlation through additional output nodes. Furthermore, various techniques for reducing the parameters of a fully connected layer have been proposed. They replace this layer with an Adaptive Fastfood transform with non-linearity [61] and a global average pooling layer [42, 59]. In CNNs, pruning is mainly performed by deleting redundant filters from convolutional layers and parameters of fully connected layers to reduce the storage and computational overhead of the network [13]. A new method for deleting redundant connections is proposed in [19]. It consists of two iterative steps, namely: *(i) pruning*, where redundant connections are deleted, and *(ii) splicing*, where deleted connections that are considered important are recovered. Also, the authors of [40] present an approach for pruning the convolutional layer filters having a low ranking, computed according to their $L1$-norm, and weakly affecting the model accuracy. They also show that the resulting model is fine-tuned/re-trained. In [50], the authors propose an approach that prunes and fine-tunes a network until a trade-off between accuracy and model size is obtained. Instead, the authors of [22] propose an approach that identifies the most relevant filters using the lasso regression method; then, it prunes irrelevant filters and reconstructs the output using unpruned filters and applying linear least squares. The inter-filter and intra-filter redundancy is investigated in [44], where the authors convert the operations of the convolutional layer into sparse matrix multiplication to process by means of a new efficient algorithm. In [71], the authors introduce a method based on gradual pruning of small magnitude weights in the training phase. Finally, the authors of [9] propose a method for pruning convolutional layers, which differs from the previous works of neuron, weight or filter pruning. Specifically, it identifies redundant connections based on the features learned in the convolutional layers; then, it prunes the involved layers.

As for quantization methods, they can be used during the training process, or after it, for an efficient inference [13]. In this context, the authors of [10] propose to use a hash function for randomly grouping weight connections into buckets; all the connections falling in the same bucket have the same weight; obtained weights are fine-tuned during the training process. In several studies, network weights are binary values (e.g., +1 and -1) during the training forward and backward phases [14] in such a way as to substitute multiply-accumulate operations with only accumulations. Instead, with quantized backpropagation [43], network weights fall in the ranges [-1, 0] and [0, +1]; in this way, multiplications in the forward and backward steps are avoided. An approach to also binarize activations is proposed in [29], whereas the authors of [26, 25] examine the effects of binarization and ternarization on the loss. In [70], the authors propose not only the quantization of weights and activations, but also the stochastic

quantization of gradients. The authors of [41] apply singular value decomposition on the filters of binarized CNNs to reduce both the parameters and the dimension of a network. In [20], the authors apply a new method on the pre-trained model; first it deletes redundant weight connections; then, it performs weight quantization using k-means; afterwards, it applies a re-training step to improve accuracy; finally, it uses the Huffman coding on the quantized model to decrease its size. In [5], the authors introduce pruning at different granularity levels and scales. In order to detect candidates for pruning, they use particle filtering, where the misclassification rate affects the weight of configurations. To further lower the model size, they adopt a fixed-point optimization.

As for knowledge distillation, the authors of [23] prove that the knowledge acquired by a large model, for which the extraction of features is easy, can be moved on a smaller model in order to facilitate the deployment operation. Specifically, they adopt a temperature for creating the soft output from the teacher model. Then, they use this temperature for training the student model from the teacher one with the objective of minimizing the error between the outputs of the two models. The authors of [52] propose a new method for training deep neural networks. Here, the training of the student model is driven by the middle layer of the teacher model, which is used not only for outputs but also to increase the accuracy of the student model. In [35], the authors paraphrase the knowledge of the teacher model into a simpler form using convolution operations involving the paraphraser and the translator. This knowledge is then moved on a student model and allows the latter to learn more easily the knowledge from the teacher model. The authors of [57] propose to match the gradients, instead of the soft outputs, for moving the knowledge from the teacher model to the student one. In [51], the authors propose an approach that generates a student model with low-precision (quantization) from the knowledge of the teacher one. Then, it uses stochastic gradient descend for optimizing the quantized elements in order to improve fitting with the teacher model. The authors of [38] introduce *On-the-fly Native Ensemble* (ONE). This method is first trained by creating a multi-branch variant of the target neural network through the addition of auxiliary branches. Then, it generates the teacher model as an ensemble of all the branches; each branch is trained using two loss terms, namely softmax cross-entropy loss and distillation loss. In [63], the information flow through a neural network is captured by a new representation called relational graph. The authors prove that the clustering coefficient and the average path length of this graph affect the neural network's predictive performance. Thus, there is a *sweet spot* of relational graphs leading to neural networks with an improved performance. Finally, in [1] the authors represent a social network graph as an artificial neural network and, then, explore the internal dynamics of the latter.

As far as the low-rank factorization is concerned, the authors of [53] propose to factorize the weight matrix of the final weight layer, of size $m \times n$ and rank r, in two matrices of size $m \times r$ and $n \times r$, to decrease the number of parameters of a deep neural network by a factor p such that $p > \frac{r \cdot (m+n)}{m \cdot n}$. In order to decrease the number of computations in both the convolutional and the fully connected layers, the authors of [16] adopt a low-rank approximation; they obtain a noticeable reduction in terms of memory usage for the weights in both convolutional and dense layers. In order to decrease the number of parameters of a CNN, the authors of [32] use singular value decomposition which decompose a tensor for speeding-up the deep neural network. The authors of [36] introduce a new approach for compressing the network model; it performs rank selection, low-rank tensor decomposition and fine-tuning and minimizes the tensor's reconstruction error. Low-rank decomposition of filters

learned from scratch during the training, instead of pre-training, is proposed in [31, 2]. Also, in [69], the authors introduce a new method for compressing and accelerating very deep CNNs; it performs network approximation in terms of non-linear units, instead of linear responses or filters. Generalized singular value decomposition, instead of stochastic gradient descend, has been used for solving non-linear problems. The authors of [39] propose a constrained-based optimization approach for detecting an optimal low-rank approximation of a trained CNN. The adopted constraints are the number of multiply-accumulate operations and the memory usage of the model.

In the past literature, different approaches have been introduced for interpreting deep learning models [62]. Some of them provide an explanation of the deep networks through the visualization and the localized inspection of high-level representations of graphs. In particular, the authors of [33] present ActiVis, which is able to visualize how neurons are activated by user-specified instances, or instance subsets, for explaining how a model generates its predictions. They provide a graph-based representation of the model allowing the local inspections of the activations at each layer codified as a node. The authors of [24] propose SUMMIT, an interactive tool aiming to detect relevant neurons and their relationships in the network. SUMMIT is based on an attribution graph that represents and summarizes important neuron connections and network substructures determining the model's outcome. In [65], the authors perform the interpretation of a CNN through the generation of an explanatory graphs representing the hidden knowledge embedded in it. Each node of the graph corresponds to a part pattern codified in a filter, while each edge maps co-activation and spatial relationships between patterns. In [67], the authors propose to use a decision tree to interpret the role of filters in a convolutional layer for prediction; it also determines at which extent filters contribute to prediction and which object parts mainly affect the latter. The authors of [66] propose a method to build a model for hierarchical object recognition based on the semantics hidden in the convolutional layers of a pre-trained CNN. This method extracts an interpretable And-Or graph with four layers in order to explain the semantic hierarchy hidden in a CNN. In [46], the authors propose a visual analytics system called CNNVis, aiming to interpret and understand a CNN model. CNNVis uses a direct acyclic graph to explore the internal learned representations in terms of multiple facets of the neurons, their features and their interactions.

3 Mapping a Convolutional Neural Network into a multilayer network

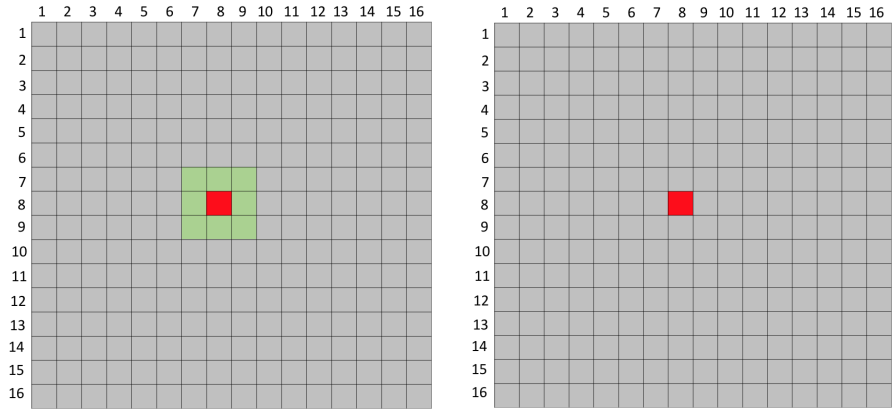
In this section, we describe our approach for mapping a CNN into a multilayer network, which represents the first main contribution of this paper. The multilayer network thus obtained can be employed to represent, analyze and manipulate the corresponding CNN.

3.1 Class network definition

In this subsection, we provide a description of a CNN in terms of a single-layer network, called *class network*. Formally, a *class network* is a weighted directed graph $G = (V, E, W)$, where V is the set of nodes, E is the set of arcs, and W is the set of arc weights.

3.1.1 Node definition

A CNN consists of M convolutional layers, each characterized by a number x of filters (also called "kernels"). In a convolutional layer, each filter slides over the input with a given stride to create a feature map. The input $\mathcal{I}$ is the original image for the first convolutional layer, or a feature map for the next convolutional layers. The application of a filter at the position $(i,\ j)$ of $\mathcal{I}$ (hereafter, $\mathcal{I}(i,j)$) provides a new element $\mathcal{O}(i,j)$ of the output feature map $\mathcal{O}$. Figure 1(a) shows the application of a filter of size 3×3 at position $\mathcal{I}(8,8)$, whereas Figure 1(b) shows the new produced element $\mathcal{O}(8,8)$ (red colored). The area of the filter is green colored.



(a) (b)

Figure 1: (a) Application of a filter of size 3×3 (green colored) to $\mathcal{I}(8,8)$; (b) the new produced element $\mathcal{O}(8,8)$ (red colored)

From all aforementioned, the set V of the nodes of G consists of a set $\{V_1, V_2, ..., V_M\}$ of node subsets. Here, the subset V_k denotes the contribution of the k^{th} convolutional layer c_k, obtained by applying the x_k filters of this layer to the input of c_k. As a consequence, a node $p \in V_k$ represents the output obtained by applying the x_k filters of c_k at some position $(i,\ j)$ of the input.

3.1.2 Arc definition

Since the application of a filter at $\mathcal{I}(i,j)$ generates a new element $\mathcal{O}(i,j)$ (see Figure 1), a direct connection between $\mathcal{I}(i,j)$ and $\mathcal{O}(i,j)$ is straightforward. Actually, in order to keep the context information, a direct connection is provided not only between $\mathcal{I}(i,j)$ and $\mathcal{O}(i,j)$, but also between $\mathcal{I}(i,j)$ and each element adjacent to $\mathcal{O}(i,j)$ within the filter area. Figure 2 shows the direct connections between $\mathcal{I}(8,8)$, on which a filter of size 3×3 is applied, and the elements $\mathcal{O}(8+a, 8+b)$, $-1 \leq a \leq 1$ and $-1 \leq b \leq 1$.

Considering the set of x_k filters of a convolutional layer c_k, there are x_k sets of direct connections (like the ones of Figure 2) between $\mathcal{I}(i,j)$ and $\mathcal{O}_k(i,j)$, one for each filter x_k. In particular, Figure 3 shows the direct connections between $\mathcal{I}(8,8)$, where three different filters of size 3×3 are applied,

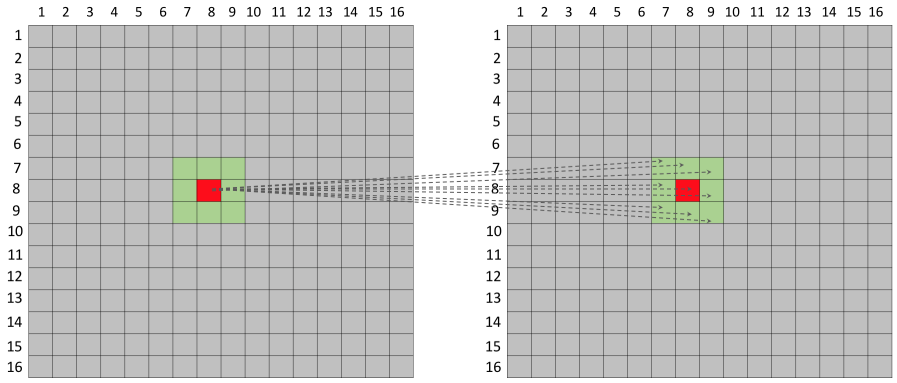


Figure 2: Direct connections (dashed lines) between $\mathcal{I}(8,8)$ and the elements $\mathcal{O}(8+a,8+b)$, $-1 \leq a \leq 1$ and $-1 \leq b \leq 1$ (red and green colored)

the new produced elements $\mathcal{O}_1(8,8)$, $\mathcal{O}_2(8,8)$ and $\mathcal{O}_3(8,8)$ of the three feature maps, and their eight neighbors at positions $\mathcal{O}_h(8+a,8+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$ and $1 \leq h \leq 3$ of each feature map.

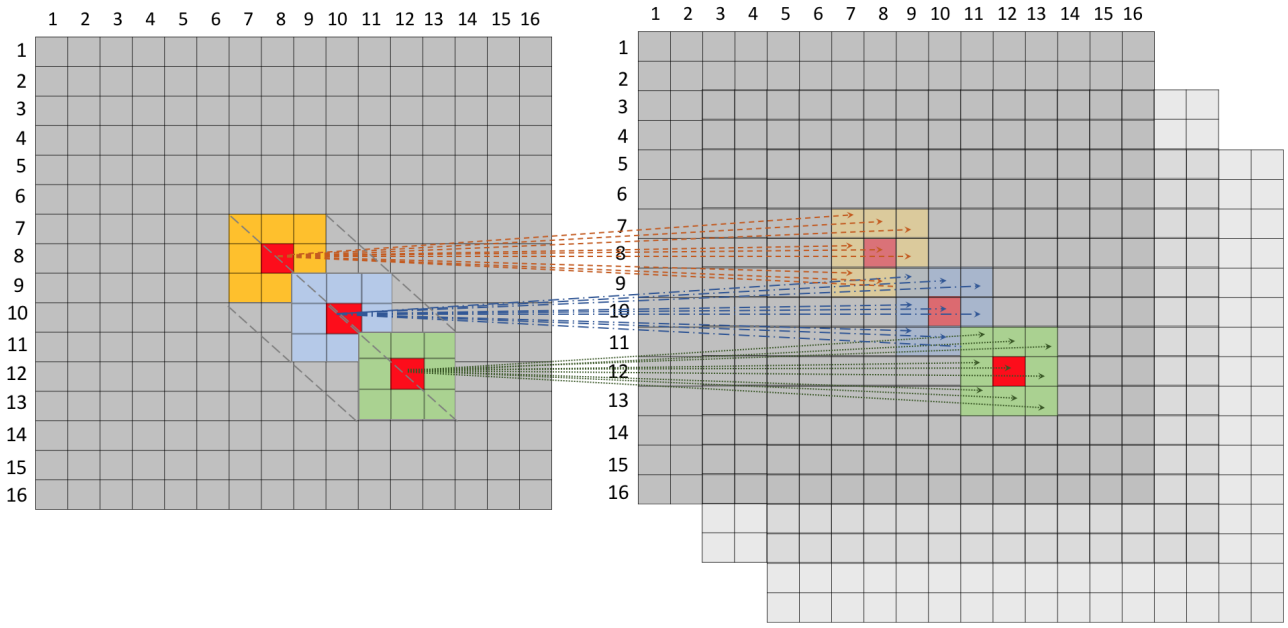


Figure 3: Direct connections (dashed lines) between $\mathcal{I}(8,8)$ and $\mathcal{O}_h(8+a,8+b)$, $-1 \leq a \leq 1$ and $-1 \leq b \leq 1$, $1 \leq h \leq 3$ obtained by applying three filters (yellow, blue and green colored)

Since the set of x_k filters is applied to the input with a given stride, a set of similar connections towards the feature maps is generated for different positions of the input.

As for the pooling layer, it shrinks the input feature maps and, in our approach, leads to the increase of the number of connections between the input and the output. This is accomplished by

sliding the filter over the input with a given stride and selecting the maximum value for each filter window. Since the maximum values are anyway elements of the feature map provided in input, the next application of a convolutional layer to the feature map returned by the max-pooling layer generates connections between the maximum values and the elements of the feature map returned by the convolutional layer. Specifically, direct connections exist between the maximum values of the feature map provided in input to the max-pooling layer and the adjacent elements of the feature map generated by the next convolutional layer.

Figure 4 shows a sample procedure of direct connection generation for a pooling layer. On the left, a pooling filter of size 2×2 and stride 2 is applied to the input; the selected maximum values are visible in the feature map as colored elements. On the right, the next application of a convolutional filter of size 3×3 at position $\mathcal{I}(2,2)$ generates 9 direct connections between $\mathcal{I}(2,2)$ and $\mathcal{O}(2+a, 2+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$.

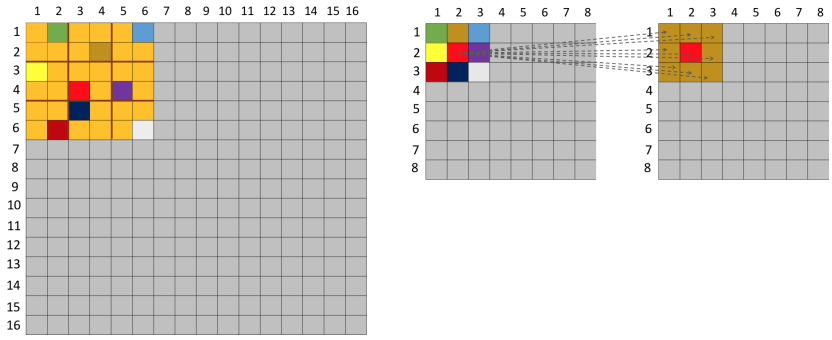


Figure 4: Example of direct connection generation for a pooling layer. The filter is of size 2×2 with stride 2. For each filter application, a colored element corresponds to the maximum value. At right, the 9 direct connections between the maximum value at position $\mathcal{I}(2,2)$ and the elements $\mathcal{O}(2+a, 2+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$ are represented by dashed lines.

3.1.3 Weight definition

The application of a filter to the element $\mathcal{I}(i,j)$ generates a new element $\mathcal{O}(i,j)$, whose value is given by the following convolution operation:

$$g(i,j) = f(i,j) * \mathcal{I}(i,j) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(s,t)\mathcal{I}(i+s, j+t), \tag{1}$$

where f is the filter of size $(2a+1) \times (2b+1)$.

Accordingly, the direct connections generated between $\mathcal{I}(i,j)$ and $\mathcal{O}(i+s, j+t)$, $-a \leq s \leq a$, $-b \leq t \leq b$ are labeled with the same weight $g(i,j)$, which is the convolution result. Figure 5 shows that the application of a filter of size 3×3 to $\mathcal{I}(8,8)$ returns $\mathcal{O}(8+a, 8+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$. The weight is the convolution result $g(8,8) = f(8,8) * \mathcal{I}(8,8) = (0 \cdot 7) + (-1 \cdot 6) + (0 \cdot 6) + (-1 \cdot 4) + (5 \cdot 4) + (-1 \cdot 2) + (0 \cdot 3) + (-1 \cdot 3) + (0 \cdot 7) = 5$.

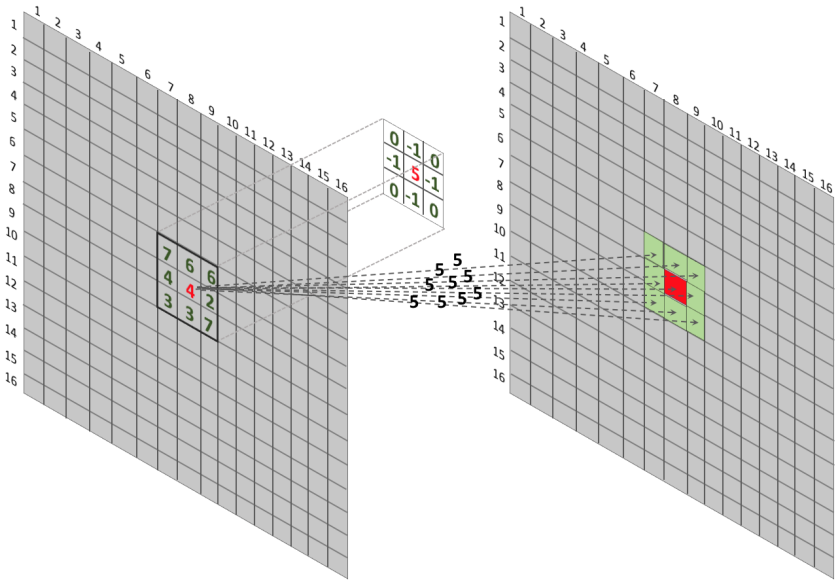


Figure 5: Application of a filter of size 3×3 to $\mathcal{I}(8,8)$ (red colored) and computation of the weights for the direct connections between $\mathcal{I}(8,8)$ and $\mathcal{O}(8+a, 8+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$

For a number x of filters, the direct connections between $\mathcal{I}(i,j)$ and $\mathcal{O}_h(i+a, j+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$, $1 \leq h \leq x$ are weighted with the values of the corresponding convolution results $g_1(i,j), g_2(i,j), \ldots, g_x(i,j)$. Figure 6 shows the application of three filters of size 3×3 (green, blue and yellow colored, respectively) to $\mathcal{I}(8,8)$. It also reports, for each filter f_h, the weighted direct connections generated between $\mathcal{I}(8,8)$ and $\mathcal{O}_h(8+a, 8+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$, $1 \leq h \leq 3$.

The three weights are the following convolution results: $g_1(8,8) = f_1(8,8) * \mathcal{I}(8,8) = 5$; $g_2(8,8) = f_2(8,8) * \mathcal{I}(8,8) = -2$; $g_3(8,8) = f_3(8,8) * \mathcal{I}(8,8) = 11$.

In order to generate the arcs of the graph G from the weights of the direct connections of the x filters, we adopt some statistical descriptors. The ultimate goal is having only one set of arcs from the node corresponding to $\mathcal{I}(i,j)$ to the node corresponding to $\mathcal{O}(i+s, j+t)$, $-a \leq s \leq a$, $-b \leq t \leq b$[1]. The weight of an arc from $\mathcal{I}(i,j)$ to $\mathcal{O}(i+s, j+t)$ is obtained by applying a suitable descriptor parameter to the weights of $g_h(i,j)$, $1 \leq h \leq x$.

The two statistical descriptors used in this context are: *(i)* the mean, and *(ii)* the median, which revealed to achieve the best performance results. Accordingly, the weight of the direct connections from $\mathcal{I}(i,j)$ to $\mathcal{O}(i+s, j+t)$ can be obtained as:

$$g_{mean}(i,j) = \frac{\sum_{h=1}^{x} g_h(i,j)}{x}, \tag{2}$$

$$g_{median}(i,j) = \begin{cases} g_{\left[\frac{x}{2}\right]}(i,j) & \text{if } x \text{ even} \\ \frac{g_{\left[\frac{x-1}{2}\right]}(i,j) + g_{\left[\frac{x+1}{2}\right]}(i,j)}{2} & \text{if } x \text{ odd} \end{cases} \tag{3}$$

[1]Here and in the following, we will use the symbols $\mathcal{I}(i,j)$ and $\mathcal{O}(i,j)$ to denote both the elements of the feature maps and the corresponding nodes of the class network.

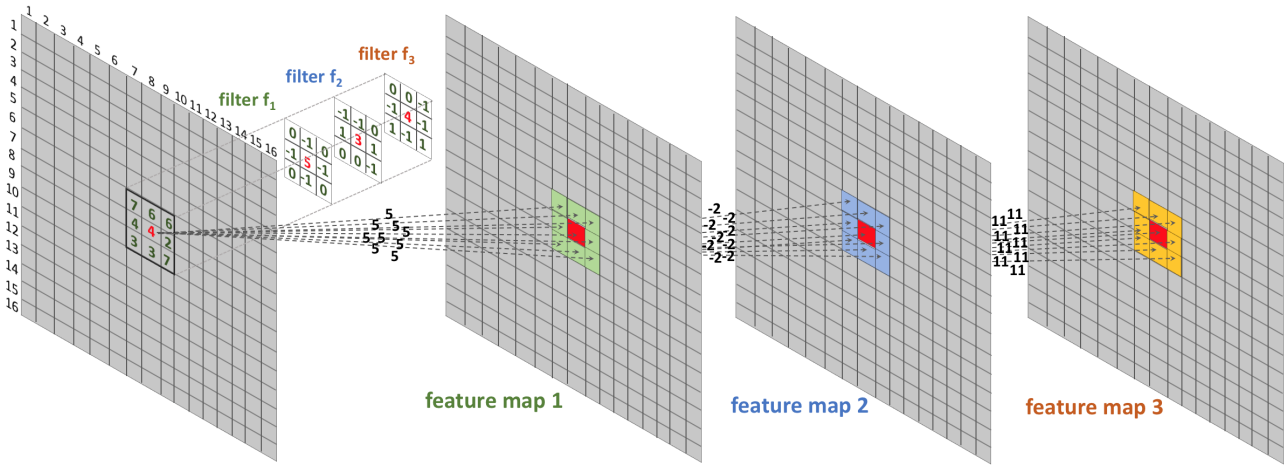


Figure 6: Application of three filters of size 3×3 (green, blue and yellow colored, respectively) to $\mathcal{I}(8,8)$ and computation, for each filter, of the weights for the direct connections between $\mathcal{I}(8,8)$ and $\mathcal{O}_h(8+a, 8+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$, $1 \leq h \leq 3$

where $g_{mean}(i,j)$ is the mean value, and $g_{median}(i,j)$ is the median value.

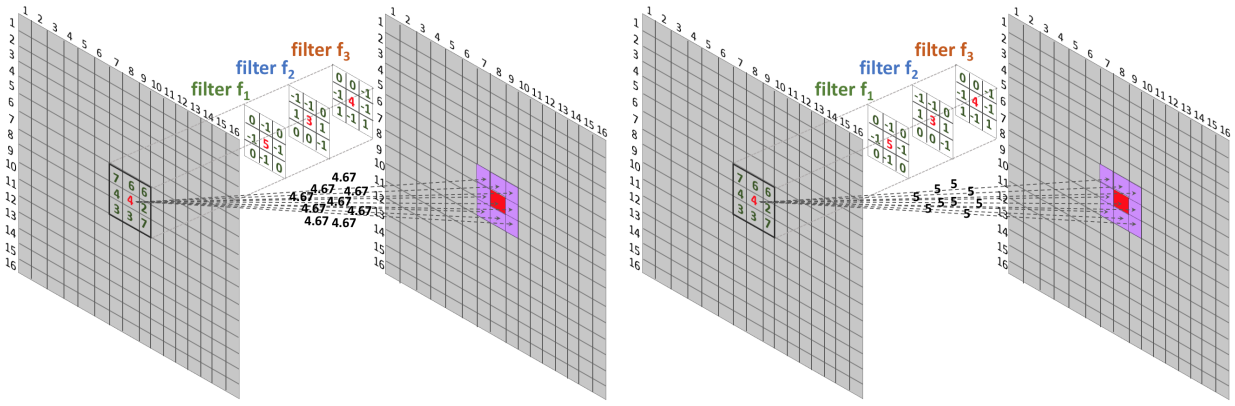
Figure 7 shows the direct connections between $\mathcal{I}(8,8)$ and $\mathcal{O}(8+a, 8+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$, whose weights are the mean value (see Figure 7(a)) and the median value (see Figure 7(b)) of the connections for the three filters depicted in Figure 6. The mean value is computed as $\frac{5-2+11}{3} = 4.67$, whereas the median value between 5, -2 and 11 is equal to 5.

From all aforementioned, the set E of the arcs of G is a set of subsets $E = \{E_1, E_2, ..., E_{M-1}\}$. Here, E_k denotes the set of arcs connecting nodes of V_k to nodes of V_{k+1}. Analogously, the set W of the weights of G consists of a set of subsets $W = \{W_1, W_2, ..., W_{M-1}\}$, where W_k is the set of weights associated with the arcs of E_k.

3.2 Mapping a CNN into a multilayer network

In the previous subsection, we have illustrated how it is possible to construct a class network representing a CNN. In this section, we show how, starting from the class network and a dataset on which the corresponding CNN should be trained and tested, it is possible to construct a more complex structure called multilayer network.

Roughly speaking, a multilayer network is a set of t class networks, one for each target class of the dataset. Formally speaking, given a dataset D consisting of a t target classes $Cl_1, Cl_2, \ldots, Cl_t$, and given a Convolutional Neural Network *cnn*, the multilayer network $\mathcal{G} = \{G^1, G^2, ...G^t\}$ corresponding to *cnn* is a set of t class networks. The class network G^h, $1 \leq h \leq t$, corresponds to the h^{th} target class of D. Figure 8 shows a sample multilayer network $\mathcal{G}$ characterized by three layers, each corresponding to a generated class network. The top (resp., middle, bottom) class network is denoted as G^1 (resp., G^2, G^3). Figure 9 shows the generation of a portion of the multilayer network $\mathcal{G}$ shown in Figure 8. It is obtained by extracting the feature maps of three target classes from *cnn*. In this last network a bank of filters of size 3×3 with stride 1, sliding over positions $(3,2)$ and $(4,2)$ of the feature maps,



(a) mean (b) median

Figure 7: Weights of the arcs from $\mathcal{I}(8,8)$ to $\mathcal{O}(8+a,8+b)$, $-1 \leq a \leq 1$, $-1 \leq b \leq 1$, obtained by applying the mean (on the left) and the median (on the right) as statistical descriptors of the corresponding direct connections

generates some arcs for the class networks of $\mathcal{G}$. The weights of these arcs are computed as described in Section 3.1.3. In particular, the mean statistical descriptor is applied. Specifically, for the class network G^1 (resp., G^2, G^3), two sets of arcs of weights 5 (resp., 1, 3) and 4 (resp., 7, 6) are generated between the first and second feature maps, and the set of arcs of weights 2 (resp., -2, 4) and 8 (resp., 9, -1) are generated between the second and the third feature maps.

3.2.1 Algorithm for building the multilayer network

In this section, we illustrate our algorithm to construct a multilayer network from a CNN and a dataset D consisting of a set of t target classes. It consists of two steps, namely: *(i)* creation of a list of patch lists, which represents a support data structure for the next step; *(ii)* creation of the multilayer network from the list of patch lists. We have defined an appropriate function for each of these steps.

The function corresponding to the first step, called `CREATE_PATCHES`, is reported in Algorithm 1. It receives a Convolutional Neural Network *cnn* and a target class Cl_h, $1 \leq h \leq t$, and constructs a list of patch lists. `CREATE_PATCHES` operates on the feature maps provided in input to each convolutional layer of *cnn*. A patch is a part of a feature map; in particular, it has the same size as the filters applied by the next convolutional layer and will give rise to a node in the multilayer network.

`CREATE_PATCHES` proceeds as follows. It uses a list *conv_layers* that initially contains the sequence of the convolutional layers present in the reference CNN. It iterates over all elements of *conv_layers*, providing in input the images of Cl_h. During each iteration, it takes the current element as the *source* and the next element as the *target*. The feature map returned as output from *source* represents the input to *target*; the latter receives this input and processes it as specified below.

At the beginning of each iteration, `CREATE_PATCHES` determines the starting and ending points of the output of *source*. In particular, the starting (resp., ending) points img_{ws} (resp., img_{we}) and img_{hs} (resp., img_{he}) represent the center of the first (resp., last) application of the convolutional filters

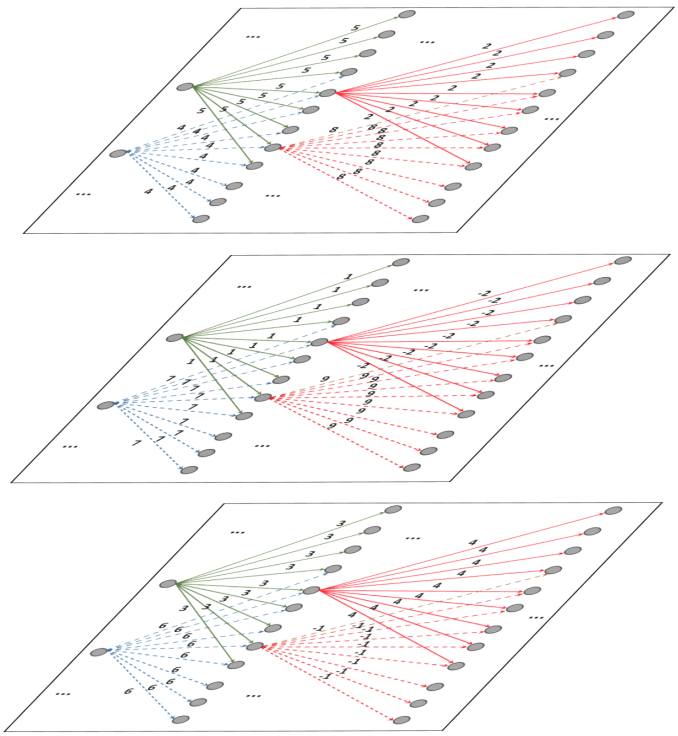


Figure 8: Sample multilayer network $\mathcal{G}$ composed of three layers corresponding to class networks G^1 (top), G^2 (middle), and G^3 (bottom).

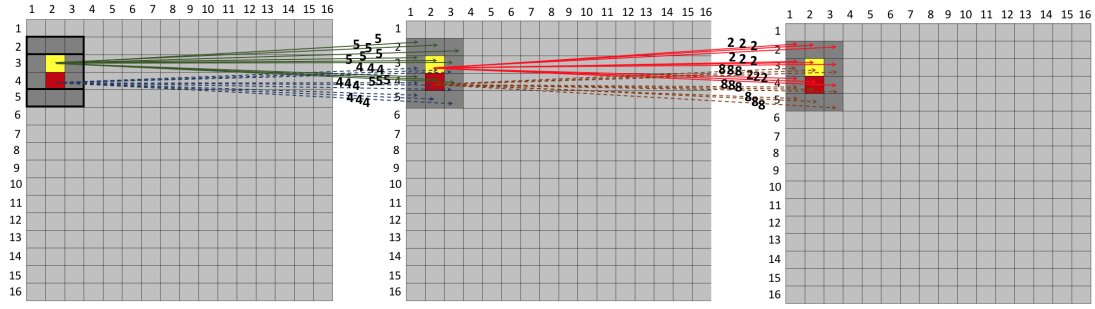
applied by *target* on the output of *source*. After this, `CREATE_PATCHES` iterates on the output of *source* and creates a patch for each application of the convolutional filter on an element. For each patch it stores its identifier, the coordinates of its center, its width and height expressed in pixels, and the corresponding source and target. At the end of the iteration, it stores the patches corresponding to a convolutional layer in a list called *patch_list*.

The lists corresponding to all the convolutional layers of *cnn* are stored in a list of lists called *list_of_patch_lists*, which represents the output of `CREATE_PATCHES`.

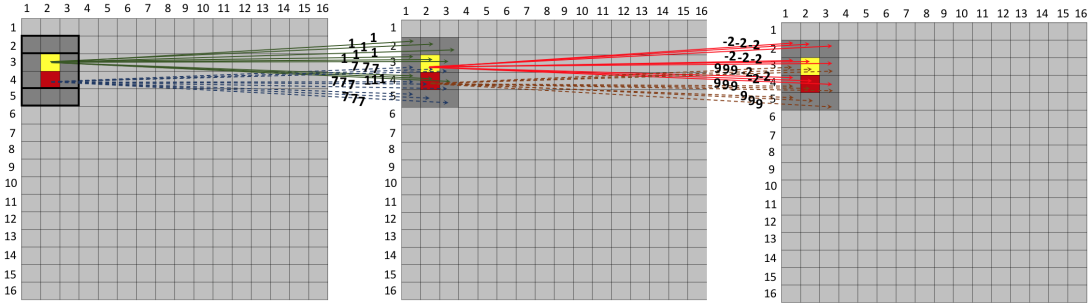
The function corresponding to the second step, called `CREATE_LAYER_NETWORK`, is shown in Algorithm 2. It receives a Convolutional Neural Network *cnn* and a target class Cl_h, $1 \leq h \leq t$, and returns a layer (specifically, the h^{th} layer G^h) of the multilayer network $\mathcal{G}$.

First, `CREATE_LAYER_NETWORK` calls the function `CREATE_PATCHES`, described in Algorithm 1, which returns the list of patch lists corresponding to *cnn* when trained with the images of the target class Cl_h. Then, it creates an initially empty network G^h. Afterwards, it iterates over the list of patch lists returned by `CREATE_PATCHES` considering two lists at a time. In particular, it considers the current list as *source* and the next one as *target*.

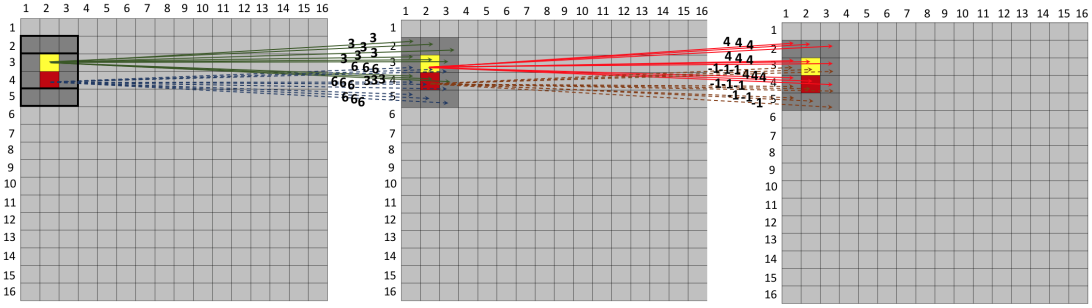
At the beginning of each iteration, `CREATE_LAYER_NETWORK` adds to G^h a node for each patch present in *source* and a node for each patch present in *target*, if they are not already present in G_h. Then, it computes a pair of parameters called w_{ratio} and h_{ratio}. In fact, as we have seen in Section 3.1.2, a pooling layer can exist between two consecutive convolutional layers, which reduces the image



(a) Class network G^1



(b) Class network G^2



(c) Class network G^3

Figure 9: Generation of a portion of the class networks G^1, G^2 and G^3 corresponding to the layers of the multilayer network $\mathcal{G}$ depicted in Figure 8

size. If this happens, the area covered by a filter in *target* is greater than that covered in *source* (see Figure 4). The parameters w_{ratio} and h_{ratio} allow us to model this phenomenon, as we will see below.

At this point, `CREATE_LAYER_NETWORK` stores in w_{bound} (resp., h_{bound}) half the width (resp., height) in pixels of the filter associated with the convolutional layer. Indeed, as we have seen in Section 3.1.2, the application of convolutional filters is done with reference to the center of the filter, and therefore of the patch. For this reason, `CREATE_LAYER_NETWORK` iterates over the source and target nodes on which the filter acts and whose coordinates are determined from those of the center of the filter, its width and its height. More specifically, given a node $node_t$ with coordinates $(node_{t_x}, node_{t_y})$, `CREATE_LAYER_NETWORK` considers all the nodes of *source* that can be processed by the filter whose

Algorithm 1 Function CREATE_PATCHES

Input

- *cnn*: a Convolutional Neural Network
- Cl_h: a target class
- *get_convolutional_layers*: a function that receives a CNN and returns the list of its convolutional layers

Output

- *list_of_patch_lists*: the list of the patch lists

```
1: function CREATE_PATCHES()
2:     list_of_patch_lists = ∅
3:     conv_layers = get_convolutional_layers(cnn)
4:     for i = 0 to len(conv_layers)-1 do
5:         patch_list = ∅
6:         source = conv_layers[i]
7:         target = conv_layers[i + 1]
8:         img_ws = ⌊target["filter_width"] / 2⌋
9:         img_hs = ⌊target["filter_height"] / 2⌋
10:        img_we = source["width"] − img_ws
11:        img_he = source["height"] − img_hs
12:        for i = img_ws to img_we do
13:            for j = img_hs to img_he do
14:                patch = (id, center, width, height, source, target)
15:                Add patch to patch_list
16:        Add patch_list to list_of_patch_lists
17:    return list_of_patch_lists
```

center falls into the rectangle defined by the coordinates of the patch of $node_t$ (this rectangle is determined thanks to w_{bound} and h_{bound}) and, for each of them, adds an arc from it to $node_t$ in G^h.

Once this arc has been inserted, `CREATE_LAYER_NETWORK` computes the corresponding weights by applying the formulas seen in Section 3.1.2. For this purpose, it first considers the feature map of *source* and selects the portion of this map relative to the inserted arcs. This portion consists of a rectangle comprised between the top left corner $(node_{s_x} - w_{bound}, node_{s_y} - h_{bound})$ and the bottom right corner $(node_{s_x} + w_{bound}, node_{s_y} + h_{bound})$. These two pairs of coordinates correspond exactly to the ones of the application of a filter to the patch of $node_s$, whose output is connected to $node_t$. Note that, for each arc, both the weight based on the mean (see Equation 2) and the one based on the median (see Equation 3) are stored.

At the end of its iterations, `CREATE_LAYER_NETWORK` has created the h^{th} layer G^h, corresponding to the the target class Cl_h, of the multilayer network $\mathcal{G}$. Applying `CREATE_LAYER_NETWORK` t times, once for each target class of the dataset D, we obtain the final multilayer network.

We end this section by pointing out that the Algorithms 1 and 2 are general and can be applied to many kinds of CNN.

Algorithm 2 Function CREATE_LAYER_NETWORK

Input

- cnn: a Convolutional Neural Network
- Cl_h: a target class
- $get\_feature\_maps$: a function that receives a CNN and a target class Cl_h and returns the list of the feature maps for each layer of cnn when trained with Cl_h

Output

- G^h: the h^{th} layer of the multilayer network $\mathcal{G}$

function CREATE_LAYER_NETWORK()
  $f\_maps = get\_feature\_maps(cnn, t)$
  $list\_of\_patch\_lists = CREATE\_PATCHES(cnn, Cl_h)$
  $G^h = \emptyset$
  for $i = 0$ to len($list\_of\_patch\_lists$)-1 **do**
    $source = list\_of\_patch\_lists[i]$
    $target = list\_of\_patch\_lists[i+1]$
    Add nodes from $source$ and $target$ to G^h if they are not present therein
    $w_{ratio} = \frac{source[\text{“}width\text{”}]}{target[\text{“}width\text{”}]}$
    $h_{ratio} = \frac{source[\text{“}height\text{”}]}{target[\text{“}height\text{”}]}$
    $w_{bound} = \left\lfloor \frac{target[\text{“}filter_{width}\text{”}]}{2} \right\rfloor$
    $h_{bound} = \left\lfloor \frac{target[\text{“}filter_{height}\text{”}]}{2} \right\rfloor$
    for $node_t$ in $target$ **do**
      $(node_{tx}, node_{ty}) = node_t[\text{“}center\_coordinates\text{”}]$
      for $node_s$ in $source$ **do**
        $(node_{s_x}, node_{s_y}) = node_s[\text{“}center\_coordinates\text{”}]$
        if $(node_{s_x} - w_{bound}) \cdot w_{ratio} \leq node_{tx} \leq (node_{s_x} + w_{bound}) \cdot w_{ratio}$ **then**
          if $(node_{s_y} - h_{bound}) \cdot h_{ratio} \leq node_{t_y} \leq (node_{s_y} + h_{bound}) \cdot h_{ratio}$ **then**
            Add an arc from $node_s$ to $node_t$ in G^h
            $map = f\_maps[source[\text{“}name\text{”}]]$
            Select the portion of f_maps starting from the top left corner $(node_{s_x} - w_{bound}, node_{s_y} - h_{bound})$ to the bottom right corner $(node_{s_x} + w_{bound}, node_{s_y} + h_{bound})$ and store it in $node_{map}$
            Add the mean and the median of $node_{map}$ as weights of the arc from $node_s$ to $node_t$
  return G^h

4 Applying the multilayer network model to compress a CNN

In the previous section, we proposed an approach to map a CNN in a multilayer network. The network thus obtained represents the tool we use to analyze and manipulate the corresponding CNN. The operations that can be performed in a CNN thanks to the multilayer network thus obtained are many and various. To give an idea of their potential, in this section, we illustrate one of them, namely the compression of a CNN, which represents the second main contribution of this paper. In particular, we first provide a description of the proposed compression approach, along with an example of its behavior. Next, we present a formalization of this approach through the use of a pseudocode algorithm.

4.1 Methodology

Consider a multilayer network $\mathcal{G}$ and let G^h be its h^{th} layer. G^h consists of a weighted directed graph. Therefore, given a node v of G^h, we can define: *(i)* the *indegree* of v as the sum of the weights of the arcs of G^h incoming into v; *(ii)* the *outdegree* of v as the sum of the weights of the arcs outgoing from v; *(iii)* the *degree* of v as the sum of its indegree and its outdegree. In the following, we use the symbol $d^h(v)$ to denote the degree of v in G^h. Instead, we use the symbol $\delta(v)$ to indicate the overall (weighted) degree of v in G^h. As we will see below, $\delta(v)$ is an important indicator of the effectiveness of the filter represented by v.

As we have seen above, a multilayer network $\mathcal{G} = \{G^1, G^2, \ldots, G^t\}$ has a layer for each target class. As a consequence, we can think that the overall degree $\delta(v)$ of the node v in $\mathcal{G}$ can be obtained by suitably aggregating the degrees $d^1(v), d^2(v), \cdots, d^t(v)$ of v in the t layers of $\mathcal{G}$. More specifically:

$$\delta(v) = \mathcal{F}(d^1(v), d^2(v), ..., d^t(v)) \tag{4}$$

where $\mathcal{F}$ is an aggregation function.

Let $d^{tot}(v) = \sum_{h=1}^{t} d^h(v)$ be the sum of the degrees of v in the t layers of $\mathcal{G}$. Our approach adopts the following entropy-based aggregation function for determining the overall degree $\delta(v)$ of v in $\mathcal{G}$ [47], [8]:

$$\delta(v) = -\sum_{h=1}^{t} \frac{d^h(v)}{d^{tot}(v)} log\left(\frac{d^h(v)}{d^{tot}(v)}\right), \tag{5}$$

This function refers to the well-known concept of *information entropy* introduced by Shannon in [54]. The definition of $\delta(v)$ in Equation 5 favors a uniform distribution of the degree of v in the different layers while it penalizes the presence of a high degree of v in few layers. The rationale underlying it is to favor those nodes whose feature extraction is balanced for different target classes [18]. Conversely, it penalizes those nodes that make a significant contribution in only few target classes.

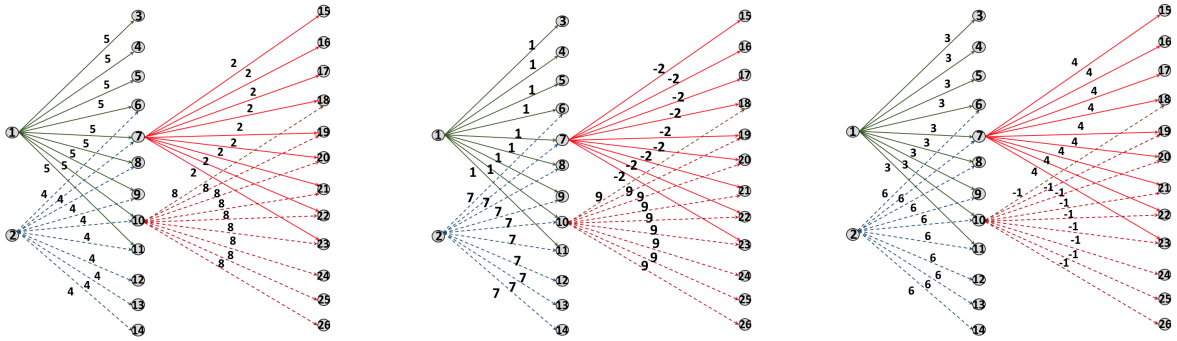
Our compression approach aims to select a subset of the nodes of $\mathcal{G}$ with the highest values of the overall degree δ. Selecting such a subset allows us to determine the best convolutional layers that will form the compressed CNN.

In particular, our approach selects the nodes of $\mathcal{G}$ whose values of overall degree δ are higher than a certain threshold:

$$th_\delta = \gamma \cdot \overline{\delta}, \tag{6}$$

Here, $\overline{\delta}$ is a statistical aggregation of the values of the overall degree δ of all nodes in $\mathcal{G}$. In particular, our approach allows the adoption of two statistical aggregators, namely mean and median. Our choice fell on these two operators because they are the ones that allowed us to achieve the best experimental results. γ is a scaling factor whose value belongs to the real interval $[0, +\infty)$. It allows us to tune the contribution of $\overline{\delta}$.

After the subset of the nodes of $\mathcal{G}$ having an overall degree δ higher than th_δ has been selected, our approach determines the set of the convolutional layers from which these nodes were extracted.



(a) Layer 1 (b) Layer 2 (c) Layer 3

Figure 10: Flattened representation of a portion of the multilayer network $\mathcal{G}$ depicted in Figure 8

Then, it creates the compressed CNN from these convolutional layers. Once this task is completed, a new training task must be performed to adjust the weights of the compressed network.

Figure 10 shows a flattened representation of a portion of the multilayer network $\mathcal{G} = \{G^1, G^2, G^3\}$ depicted in Figure 8. Here, all the nodes are numbered from 1 to 26.

In order to compute the threshold th_δ, $\overline{\delta}$ is first determined as the mean of the overall degrees δ of the nodes of $\mathcal{G}$. For a node v, $\delta(v)$ is computed according to Equation 5. For instance, $\delta(2)$ is computed as follows:

$$\begin{aligned}\delta(2) &= -\left[\frac{d^1(2)}{d^{tot}(2)} log\left(\frac{d^1(2)}{d^{tot}(2)}\right)\right] - \left[\frac{d^2(2)}{d^{tot}(2)} log\left(\frac{d^2(2)}{d^{tot}(2)}\right)\right] + \left[\frac{d^3(2)}{d^{tot}(2)} log\left(\frac{d^3(2)}{d^{tot}(2)}\right)\right] = \\ &= -\left[\frac{36}{153} \cdot log\left(\frac{36}{153}\right)\right] - \left[\frac{63}{153} \cdot log\left(\frac{63}{153}\right)\right] - \left[\frac{54}{153} \cdot log\left(\frac{54}{153}\right)\right] = \\ &= 0.34 + 0.36 + 0.37 = 1.07\end{aligned} \tag{7}$$

where $d^1(2)$, $d^2(2)$ and $d^3(2)$ are the degrees of node 2 for G^1, G^2 and G^3, respectively, whereas $d^{tot}(2) = 153$ is the total degree of this node.

Analogously, the overall degree δ of the other nodes are: $\delta(1) = \delta(3) = \delta(4) = \delta(5) = 0.94$, $\delta(6) = \delta(8) = \delta(9) = \delta(11) = 1.09$, $\delta(12) = \delta(13) = \delta(14) = 1.07$, $\delta(18) = \delta(19) = \delta(20) = \delta(21) = \delta(22) = \delta(23) = 1.00$. Finally, nodes 7, 10, 15, 16, 17, 24, 25, and 26 do not give any contribution, since the *log* in Equation 5 has no meaning for negative numbers.

Observe that $\delta(2) = 1.07 < \delta(11) = 1.09$, because the degree distribution of node 11 (i.e., $d^1(11) = 9$, $d^2(11) = 8$, and $d^3(11) = 9$), is more balanced over the three network layers than the one of node 2 (i.e., $d^1(2) = 36$, $d^2(2) = 63$, and $d^3(2) = 54$) – see Figure 10.

The value of $\overline{\delta}$, adopting the mean as statistical aggregator, is computed as follows:

$$\overline{\delta} = \frac{[(0.94 \cdot 4) + (1.07 \cdot 4) + (1.09 \cdot 4) + (1.00 \cdot 6)]}{26} = \frac{18.40}{26} = 0.71 \tag{8}$$

Let the scaling factor γ be equal to 1.50. Then, the threshold th_δ will be computed as $th_\delta = \gamma \cdot \overline{\delta} = 1.50 \cdot 0.71 = 1.06$. The nodes with degree $\delta > th_\delta$ are 2, 6, 8, 9, 11, 12, 13, 14. They are located in the

first and second feature maps (see Figures 9 and 10). Hence, the compressed CNN model consists of the first and second convolutional layers, while the third convolutional layer is pruned from the model.

4.2 Approach formalization

In this subsection, we present the formalization of the CNN compression approach that we described in the previous subsection. The corresponding pseudocode can be found within the function `COMPRESS_CNN` shown in Algorithm 3.

This function receives: *(i)* the Convolutional Neural Network *cnn* that we want to compress; *(ii)* the multilayer network $\mathcal{G}$ associated with *cnn* and computed by applying the approach described in Section 3.2; *(iii)* the scaling factor γ that we have seen in Equation 6; *(iv)* the type $aggr_{type}$ of statistical aggregation function used in the computation of $\overline{\delta}$ in Equation 6. At present, the possible types are "mean" and "median".

It also uses some support functions, namely:

- *compute_overall_degrees*, which computes the value of the overall degree δ of each node of $\mathcal{G}$.
- *get_convolutional_layers*, which receives a CNN and returns the list of its convolutional layers.
- *mean* (resp., *median*), which receives the set of overall degrees δ of the nodes of $\mathcal{G}$ and computes $\overline{\delta}$ which is their mean (resp., median).
- *remove_convolutional_layers*, which receives a Convolutional Neural Network *cnn* and a list *removable_layers* of convolutional layers to be pruned from *cnn* and returns a compressed Convolutional Neural Network $\overline{cnn}$, in which the layers of *removable_layers* have been pruned.

First, `COMPRESS_CNN` creates an empty list *preserving_layers*; it will contain all the layers of *cnn* to be preserved in $\overline{cnn}$. Then, it calls the function *compute_overall_degrees* to compute the list δ of the overall degrees of the nodes of $\mathcal{G}$. Afterwards, it computes the threshold th_δ by applying Equation 6 and taking $aggr_{type}$ into account.

At this point, for each node v of $\mathcal{G}$, `COMPRESS_CNN` checks whether its overall degree $\delta(v)$ is greater than th_δ. In the affirmative case, it adds the convolutional layer associated with v to the list *preserving_layers*. This way of proceeding implies that a convolutional layer is preserved if it has at least one node that makes a significant contribution to the operation of *cnn*.

Once all preserving layers have been identified, `COMPRESS_CNN` obtains the prunable layers by calling the function *get_convolutional_layers* with *cnn* as input and subtracting from the layers returned by it those present in the list *preserving_layers*.

After determining the layers to be pruned, `COMPRESS_CNN` calls the function *remove_convolutional_layers* giving it *cnn* and *removable_layers* as input. The latter function prunes all the layers of *removable_layers* from *cnn* thus obtaining $\overline{cnn}$, which is also the output of `COMPRESS_CNN`.

5 Experiments

In this section, we evaluate the performances of our approaches. We pointed out that they are general and can be applied to several kinds of CNN. In our experiments, we decided to apply it to VGG16

Algorithm 3 COMPRESS_CNN

Input

- cnn: a CNN to compress
- $\mathcal{G}$: the multilayer network associated with cnn
- γ: a real number representing the scaling factor in the computation of th_δ
- $aggr_{type}$: the statistical aggregation function chosen for the computation of the overall degrees
- $compute\_overall\_degrees$: a function that computes the overall degree δ of each node of $\mathcal{G}$
- $get\_convolutional\_layers$: a function that returns the list of the convolutional layers of a CNN
- $median$: a function that returns the median of a list of values
- $mean$: a function that returns the mean of a list of values
- $remove\_convolutional\_layers$: a function that prunes a list of convolutional layers from a CNN

Output

- $\overline{cnn}$: a compressed version of cnn

function COMPRESS_CNN()
  $preserving\_layers = \emptyset$
  $\delta = compute\_overall\_degrees(\mathcal{G}, aggr_{type})$
  if $aggr_{type}$ = "mean" **then**
    $\overline{\delta} = mean(\delta)$
  else if $aggr_{type}$ = "median" **then**
    $\overline{\delta} = median(\delta)$
  $th_\delta = \gamma \cdot \overline{\delta}$
  for each node v of $\mathcal{G}$ **do**
    if $\delta(v) > th_\delta$ **then**
      Add the convolutional layer associated with v to $preserving\_layers$
  $removable\_layers = get\_convolutional\_layers(cnn) \setminus preserving\_layers$
  $\overline{cnn} = remove\_convolutional\_layers(cnn, removable\_layers)$
  return $\overline{cnn}$

[55]; indeed, it is a benchmark vision CNN that won the ILSVR (ImageNet) competition in 2014. We tested the effectiveness of our approach in two well-known computer vision tasks, namely: *(i)* handwriting character recognition, and *(ii)* object recognition. In that sense, the image datasets used for the experiments are MNIST and CALTECH-101, which are considered well suited benchmarks for testing CNN architectures in these tasks [3, 4].

MNIST[2] consists of 60,000 training and 10,000 test images representing binary handwritten digits categorized into 10 target classes, which correspond to the digits from 0 to 9. Images from MNIST represent a portion of the larger NIST[3], a well-known dataset for evaluating computer vision and pattern recognition approaches. Each image of MNIST is of size 28×28 pixels in bitmap (`.bmp`) format, centered and size-normalized.

CALTECH-101[4] is a dataset with 9,146 colored pictures of different objects belonging to 101 semantic classes (e.g. chair, ant, watch, pizza). Each class comprises from 40 to 800 images. All images are in JPEG (`.jpg`) format. The original size of each image is about 300×200 pixels but, in

[2] `http://yann.lecun.com/exdb/mnist/`

[3] `https://www.nist.gov/srd/nist-special-database-19`

[4] `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`

our case, it was normalized to 128×128 pixels.

In our campaign, we first trained the VGG16 model from scratch. Then, we generated the layers of the multilayer network from the target classes of the dataset. For each target class, we computed the average of the feature maps over the image set before proceeding with the compression and use them as the weights of the arcs. As for MNIST, we used the predefined training and test sets and the predefined classes. As for CALTECH-101, we performed an holdout validation which used 80% of images for training and 20% of them for testing. Furthermore, we adopted a stratification method on the test set for keeping the same number of images per class. Finally, we employed a trial and error procedure, which determined the following best parameter values for VGG16: *(i)* batch size equal to 128 for MNIST and 64 for CALTECH-101; *(ii)* learning rate η equal to 0.0001; *(iii)* Adam optimizer; *(iv)* epoch number equal to 100; *(v)* 10% of training data for validation. For limiting the overfitting, we monitored the validation loss and stopped the training when no more changes of the validation loss occurred for three iterations.

We first performed a sensitivity analysis for studying the impact of the scaling factor γ on the compression performances of our approach when also varying the statistical aggregation function (i.e. mean and median) used in the computation of $\overline{\delta}$ (see Section 4.1). Then, we compared the results obtained by our approach with the best combination of γ and statistical aggregation function with the ones obtained by another competing method on the same datasets.

We carried out our experiments on the free version of Google Colab, which provided a GPU NVIDIA Tesla K80, 12 GB RAM, and 2 Intel Xeon CPUs 2.30GHz. The programming environment consisted of Python 3.7, Tensorflow 2.6, and NetworkX 2.6.3.

5.1 Reference Convolutional Neural Network

The VGG16 [55, 17] network is characterized by five convolutional blocks, named as $Conv_i$, $1 \leq i \leq 5$, five max pooling layers (red colored), and three fully connected layers (green colored), as shown in Figure 11. Each convolutional block consists of two or three convolutional layers with a given number of filters of size 3×3 and stride 1. One of the configurations also includes convolutional filters of size 1×1, which can be considered as a linear transformation of the input channels. Padding in the convolutional layers with filters of size 3×3 preserves the spatial resolution after the convolution and is fixed to 1 pixel. The convolutional blocks are interleaved with max pooling layers with filters of size 2×2 and stride 2. This configuration of convolutional and max pooling layers is consistent through the whole network structure. It is followed by three fully connected layers; the first two have the same number of channels, while the third one has a number of channels equal to the number of target classes. The final part corresponds to the softmax layer, which returns the output. All the hidden layers are characterized by the rectification non-linearity (ReLU).

In the standard VGG16 network architecture, i.e., the one shown in Figure 11, the input is an RGB image of size 224×224. Also, the first two fully connected layers are composed of 4096 channels, while the third one has 1000 channels, which correspond to the target classes of the ILSVRC-2012 dataset competition [55]. Finally, VGG16 has about 138 million parameters. In our specific case, the input to VGG16 can be an image with size different from 224×224. Thus, the number of parameters of our VGG16 is varied accordingly.

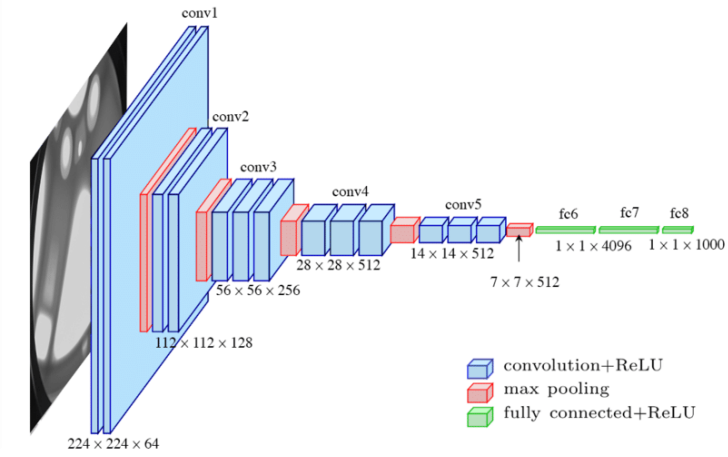


Figure 11: The VGG16 network architecture

5.2 Obtained results

5.2.1 Tuning and performances

In Figure 12 (resp. Figure 13) we show some measures obtained by VGG16 when the input is MNIST (resp., CALTECH-101). In particular, we report the values of accuracy, precision, recall, mean time per epoch (in seconds) of the CNN training phase, number of CNN parameters and number of convolutional layers pruned by our compression method. The scaling factor γ varies from 0.25 to 2.00 with steps of 0.25. This revealed as the best working range for our approach. Recall that $\gamma = 0$ corresponds to no compression.

We report the values of the above metrics when the statistical aggregation function for computing $\overline{\delta}$ is the mean or the median and when the arc weight is based on the mean or the median. As a consequence, we have four possible combinations that we represent as $(\overline{\delta}_{mean}, g_{mean})$, $(\overline{\delta}_{median}, g_{mean})$, $(\overline{\delta}_{mean}, g_{median})$ and $(\overline{\delta}_{median}, g_{median})$, respectively.

Figure 14 (resp., Figure 15) shows the VGG16 convolutional layers preserved and pruned by our compression approach for the first (resp., last) two configurations when the input is MNIST. Figures 16 and 17 show the same data when the input is CALTECH-101. In these figures, green circles denote preserved layers whereas red crosses indicate pruned ones.

Note that the accuracy, precision and recall of MNIST are higher than 0.97 for all configurations. In particular, the highest values of these parameters are obtained for the configuration $(\overline{\delta}_{median}, g_{mean})$ and γ greater than 0.75. In this case, the accuracy is higher than 0.99 whereas precision and recall are up to 0.99 when γ ranges between 1.00 and 1.50. With these configurations, our compression approach prunes the first two layers of $Conv_3$ and the first layer of $Conv_4$. Pruning also the third layer of $Conv_3$ and the first layer of $Conv_2$ leads to a slight decrease of accuracy, precision and recall. The configuration $(\overline{\delta}_{median}, g_{mean})$ guarantees the best performances but it also requires the highest mean epoch time and the highest number of parameters. It also leads to the lowest number of pruned convolutional layers that gradually increases when γ increases.

The configuration $(\overline{\delta}_{mean}, g_{mean})$ leads to lower values of accuracy, precision and recall than the

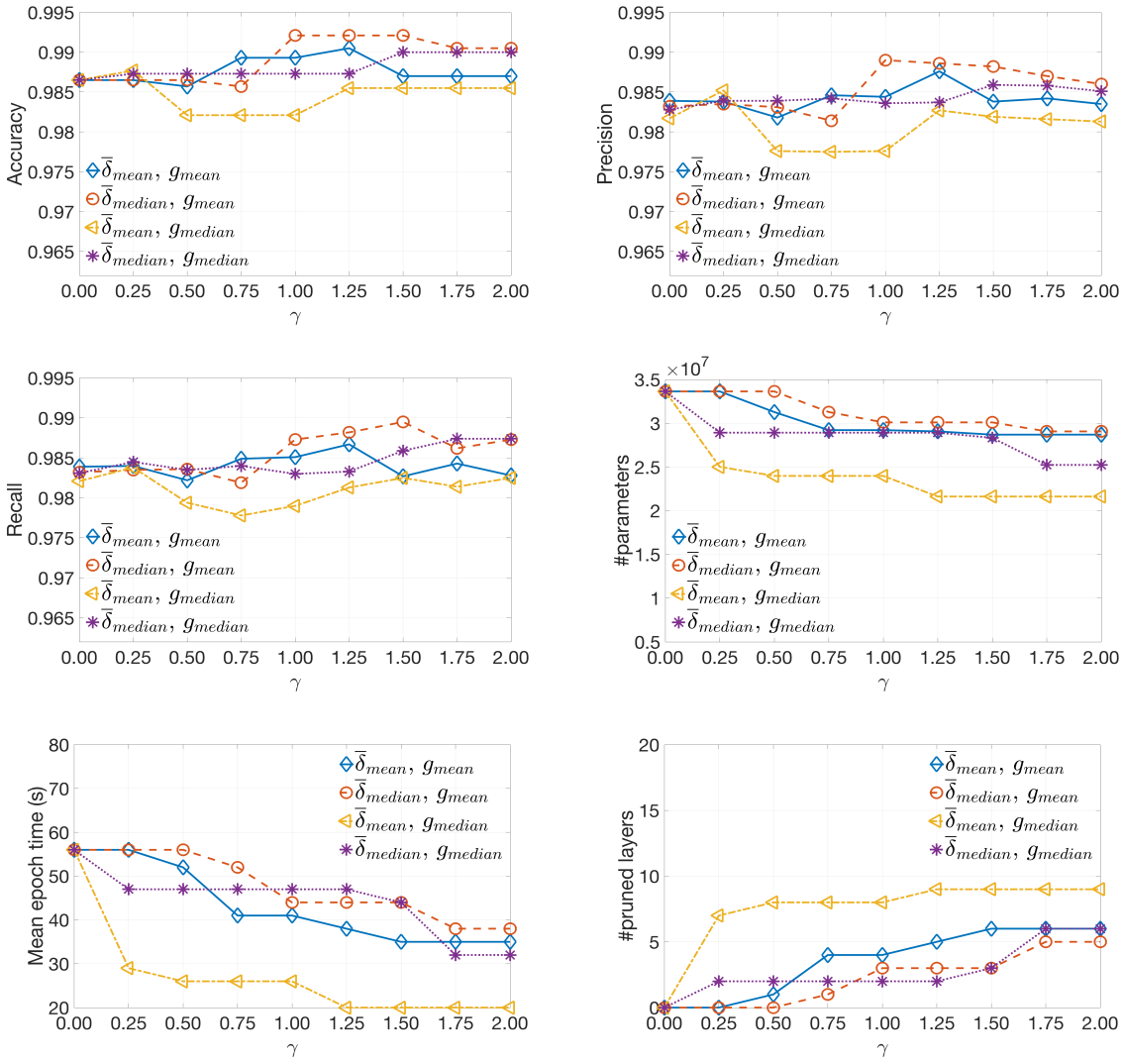


Figure 12: Accuracy, precision, recall, mean epoch time (in seconds) of the training phase, number of CNN parameters, and number of pruned convolutional layers obtained by our compression approach when MNIST dataset is given in input to VGG16

previous configuration when γ is higher than 0.75. The highest values of these parameters are obtained for $\gamma = 1.25$. When γ ranges between 0.75 and 1.25, $Conv_3$ and the first layer of $Conv_4$ are pruned. When $\gamma = 1.25$ also the first layer of $Conv_2$ can be pruned without a significant decrease of the performance values. Instead, if also the second layer of $Conv_2$ is pruned, the accuracy, precision and recall start to decrease and become lower than 0.99. If compared with $(\overline{\delta}_{median}, g_{mean})$, this configuration leads to an average decrease of the mean epoch time of 4.9s and of the number of parameters of 1.06 million. It also leads to an increase of the number of pruned convolutional layers of 1.5.

Analogous trends can be observed for the configuration $(\overline{\delta}_{median}, g_{median})$. In this case, the values

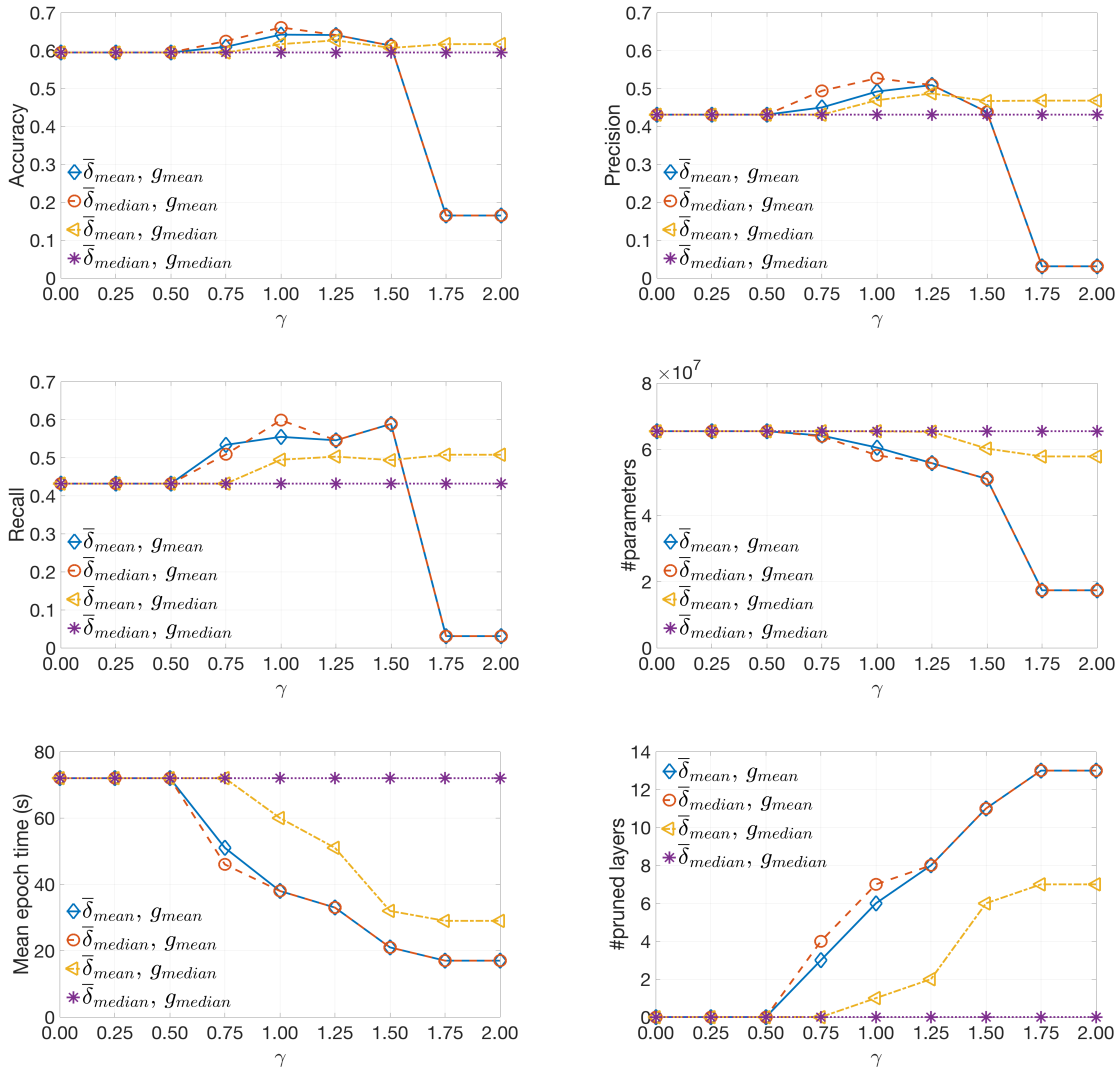


Figure 13: Accuracy, precision, recall, mean epoch time (in seconds) of the training phase, number of CNN parameters, and number of pruned convolutional layers obtained by our compression approach when CALTECH-101 is given in input to VGG16

of accuracy, precision and recall are up to 0.99 for γ between 1.50 and 2.00. This configuration leads to the pruning of the first layer of $Conv_2$ and $Conv_4$, the first two layers of $Conv_3$ and the third layer of $Conv_4$ and $Conv_5$. If compared with the configuration $(\overline{\delta}_{mean}, g_{mean})$, this one leads to an average decrease of: *(i)* the mean epoch time of 1.3s, *(ii)* the number of parameters of 1.89 million, and *(iii)* the number of pruned convolutional layers of 0.87.

Finally, as for the configuration $(\overline{\delta}_{mean}, g_{median})$, the accuracy, precision and recall reached the lowest values. For instance, the accuracy value is below 0.985 whereas precision and recall are up to 0.98 when γ ranges between 0.50 and 1.00. On the other side, this configuration leads to a rapid decrease of the mean epoch time below 30s and of the number of CNN parameters below 25 million;

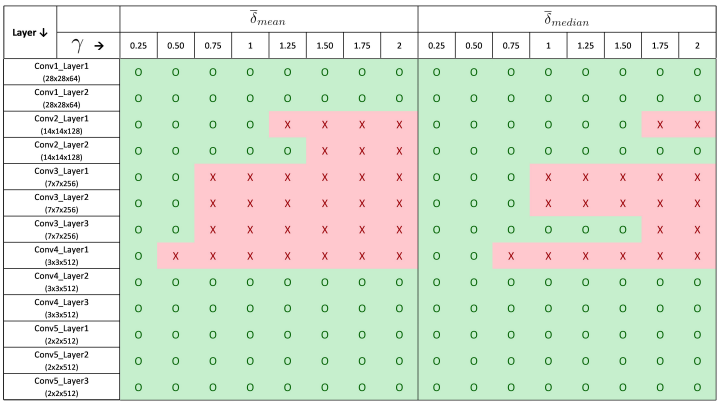


Figure 14: Convolutional layers of VGG16 preserved and pruned by our compression algorithm for MNIST (arc weights based on mean)

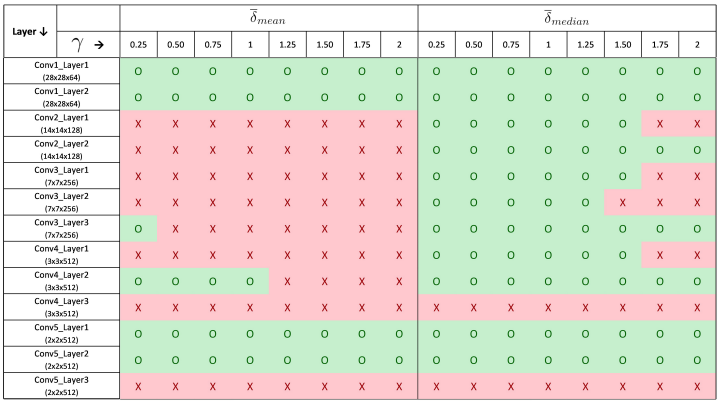
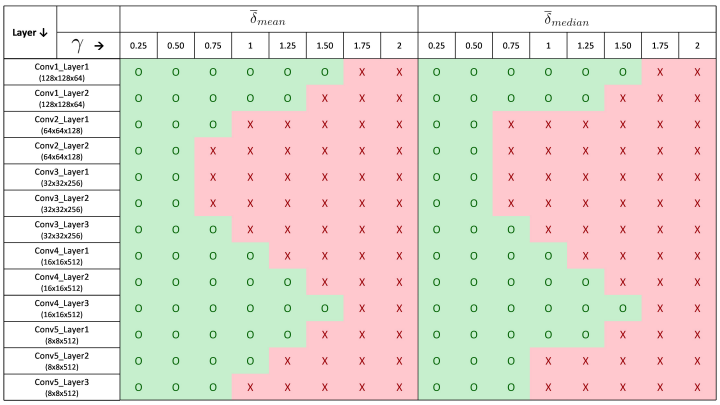


Figure 15: Convolutional layers of VGG16 preserved and pruned by our compression algorithm for MNIST (arc weights based on median)

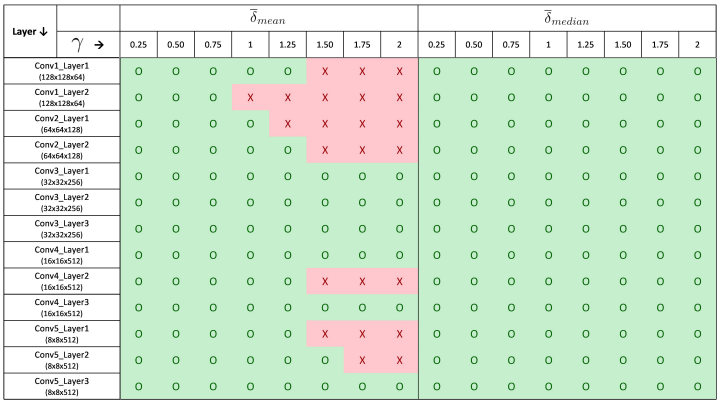
at the same time, there is an increase of the pruned layers up to 8. In this case, the compressed CNN model only preserves the whole $Conv_1$, the second layer of $Conv_4$ and the first two layers of $Conv_5$. Interestingly, pruning also the second layer of $Conv_4$ does not lead to a significant decrease of accuracy, precision and recall if γ is higher than 1. At the same time, this choice further reduces the mean epoch time and the number of CNN parameters.

As far as CALTECH-101 is concerned, the highest values of the performance measures are obtained when γ ranges from 0.75 to 1.50. More specifically, the highest value of accuracy (resp., precision, recall) is 0.661 (resp., 0.527, 0.599) and is reached when $\gamma = 1$ for the configuration $(\overline{\delta}_{median}, g_{mean})$. With this configuration, when γ ranges between 0.75 and 1.50, the mean epoch time and the number of CNN parameters gradually decrease, whereas the number of pruned convolutional layers rapidly increases. This trend can be observed in Figure 16, where the convolutional layers are rapidly pruned until the CNN model consists of the first layer of $Conv_1$ and the third layer of $Conv_4$. Interestingly,



Layer ↓	γ →	$\overline{\delta}_{mean}$								$\overline{\delta}_{median}$							
		0.25	0.50	0.75	1	1.25	1.50	1.75	2	0.25	0.50	0.75	1	1.25	1.50	1.75	2
Conv1_Layer1 (128x128x64)		o	o	o	o	o	o	x	x	o	o	o	o	o	o	x	x
Conv1_Layer2 (128x128x64)		o	o	o	o	o	x	x	x	o	o	o	o	o	x	x	x
Conv2_Layer1 (64x64x128)		o	o	o	x	x	x	x	x	o	o	x	x	x	x	x	x
Conv2_Layer2 (64x64x128)		o	o	x	x	x	x	x	x	o	o	x	x	x	x	x	x
Conv3_Layer1 (32x32x256)		o	o	x	x	x	x	x	x	o	o	x	x	x	x	x	x
Conv3_Layer2 (32x32x256)		o	o	x	x	x	x	x	x	o	o	x	x	x	x	x	x
Conv3_Layer3 (32x32x256)		o	o	o	x	x	x	x	x	o	o	o	x	x	x	x	x
Conv4_Layer1 (16x16x512)		o	o	o	o	x	x	x	x	o	o	o	o	x	x	x	x
Conv4_Layer2 (16x16x512)		o	o	o	o	o	x	x	x	o	o	o	o	o	x	x	x
Conv4_Layer3 (16x16x512)		o	o	o	o	o	o	x	x	o	o	o	o	o	o	x	x
Conv5_Layer1 (8x8x512)		o	o	o	o	o	x	x	x	o	o	o	o	o	x	x	x
Conv5_Layer2 (8x8x512)		o	o	o	o	x	x	x	x	o	o	o	x	x	x	x	x
Conv5_Layer3 (8x8x512)		o	o	o	x	x	x	x	x	o	o	o	x	x	x	x	x

Figure 16: Convolutional layers of VGG16 preserved and pruned by our compression algorithm for CALTECH-101 (arc weights based on mean)



Layer ↓	γ →	$\overline{\delta}_{mean}$								$\overline{\delta}_{median}$							
		0.25	0.50	0.75	1	1.25	1.50	1.75	2	0.25	0.50	0.75	1	1.25	1.50	1.75	2
Conv1_Layer1 (128x128x64)		o	o	o	o	o	x	x	x	o	o	o	o	o	o	o	o
Conv1_Layer2 (128x128x64)		o	o	o	x	x	x	x	x	o	o	o	o	o	o	o	o
Conv2_Layer1 (64x64x128)		o	o	o	o	x	x	x	x	o	o	o	o	o	o	o	o
Conv2_Layer2 (64x64x128)		o	o	o	o	o	x	x	x	o	o	o	o	o	o	o	o
Conv3_Layer1 (32x32x256)		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Conv3_Layer2 (32x32x256)		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Conv3_Layer3 (32x32x256)		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Conv4_Layer1 (16x16x512)		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Conv4_Layer2 (16x16x512)		o	o	o	o	o	x	x	x	o	o	o	o	o	o	o	o
Conv4_Layer3 (16x16x512)		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Conv5_Layer1 (8x8x512)		o	o	o	o	o	x	x	x	o	o	o	o	o	o	o	o
Conv5_Layer2 (8x8x512)		o	o	o	o	o	o	x	x	o	o	o	o	o	o	o	o
Conv5_Layer3 (8x8x512)		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o

Figure 17: Convolutional layers of VGG16 preserved and pruned by our compression algorithm for CALTECH-101 (arc weights based on median)

in spite of the presence of only two layers, the performance measures keep the same or higher values than the ones obtained with no compression (i.e., with $\gamma = 0$).

The configuration $(\overline{\delta}_{mean}, g_{mean})$ shows closely related trends with the configuration $(\overline{\delta}_{median}, g_{mean})$ in all measures.

By contrast, the performance measures of $(\overline{\delta}_{mean}, g_{median})$ diverge from the ones obtained by the other configurations. In fact, they show an increasing trend against γ and their value becomes even higher than the ones obtained with no compression when γ is high. On the other side, the mean epoch time is on average 12.6s higher, the number of CNN parameters is 13.5 million higher and the number of pruned convolutional layers is 4.12 less than the previous two configurations. This is justified by the reduced number of convolutional layers pruned with this configuration, which gradually comprises the two blocks $Conv_1$ and $Conv_2$, the second layer of $Conv_4$ and the first two layers of $Conv_5$.

Finally, the configuration $(\overline{\delta}_{median}, g_{median})$ does not provide any compression, and all the six

measures into consideration do not show any change against γ.

From all previous reasoning, we can conclude that the configuration that guarantees the best tradeoff between costs and benefits is $(\overline{\delta}_{mean}, g_{mean})$ with $\gamma = 1.25$. In fact, in this case, the values of the performance measures are higher than the corresponding ones without compression for both datasets. At the same time, the mean epoch time and the number of CNN parameters are acceptably low, whereas the number of pruned convolutional layers is high.

From a cross comparison of the layer distributions, the best tradeoff between costs and benefits corresponds to pruning $Conv_2$, $Conv_3$ and the first layer of $Conv_4$.

Clearly, if we are not interested to the best tradeoff, but we are willing to sacrifice costs (i.e., to accept a high mean epoch time and a high number of CNN parameters) in order to maximize benefits (i.e., high values of accuracy, precision and recall) the best configuration is $(\overline{\delta}_{median}, g_{mean})$.

5.2.2 Comparison results

In order to highlight the importance of adopting a multilayer network for supporting the representation and manipulation of CNNs, we compare our compression method with an analogous one based on a single-layer network. This approach considers each target class as a single contribution to the CNN compression. As a consequence, given a class network G^h, it first computes $\overline{\delta^h}$, $1 \leq h \leq t$, as a statistical aggregation of the values of the degree δ of all the nodes of G^h. Then, it calculates $th^h_\delta = \gamma \cdot \overline{\delta^h}$, $1 \leq h \leq t$. Afterwards, it selects the subset of the nodes of G^h, $1 \leq h \leq t$, having a degree δ higher than th^h_δ. Finally, it determines the set of nodes from which the convolutional layers of the compressed CNN are extracted by computing the intersection of these t subsets.

Table 1 shows the performance measures obtained by VGG16 when the input dataset is MNIST (resp., CALTECH-101). In particular, we report the values of accuracy, precision, recall, mean time per epoch (in seconds) of the CNN training phase, number of CNN parameters and number of convolutional layers pruned obtained by applying our multilayer network based (indicated by "m") compression approach and the single layer network based one (denoted by "s") described above. In order to make the comparison as objective as possible, we identified the best configuration also for the single-layer network based approach and adopted it in the comparison. This configuration states that the statistical aggregation function adopted in the computation of $\overline{\delta^h}$, $1 \leq h \leq t$, is the mean, the arc weight is based on the mean (g_{mean}), and $\gamma = 1.25$.

Performance measures	Accuracy		Precision		Recall		Mean epoch time (s)		#parameters		#pruned layers	
multilayer (m) single-layer (s)	m	s	m	s	m	s	m	s	m	s	m	s
MNIST	0.990	0.987	0.988	0.984	0.987	0.983	38.0	44.1	$2.9 \cdot 10^7$	$3.1 \cdot 10^7$	5	3
CALTECH-101	0.641	0.51	0.509	0.315	0.546	0.326	33.1	19.0	$5.6 \cdot 10^7$	$2.5 \cdot 10^7$	8	12

Table 1: Performance measures obtained by our compression approach based on a multilayer network (m) and a single-layer one (s) when MNIST and CALTECH-101 are provided in input

As for MNIST, we observe that the compression method based on multilayer network obtains the highest values of accuracy, precision and recall, whereas the mean epoch time is 6s lower, the number of CNN parameters is 2 million lower, and the number of pruned convolutional layers is 2 more than

the ones obtained by adopting a single-layer network.

As for CALTECH-101, we observe that the compression method based on single-layer network obtains better values of mean epoch time, number of CNN parameters and number of pruned convolutional layers than the one based on multilayer network. However, the former leads to much lower results than the latter for accuracy (0.641 against 0.510), precision (0.509 against 0.315) and recall (0.546 against 0.326).

5.3 Discussion

In this section, we draw some considerations on the approaches presented in this paper. First, we observe that our approach to map a CNN into a multilayer network is general and can be applied to most classical CNN architectures, such as LeNet, AlexNet, GoogLeNet, VGG19 and so forth.

Another advantage of our multilayer network based representation is that it provides important insights about what is happening under the hood of the CNN. In fact, it allows us to identify the best performing nodes and describe how they interact with their neighbors. We can observe how information flows through the arcs of the multilayer network, which is a representation of how the corresponding CNN filters process images. The ability of our approach to identify the most important convolutional layers of a CNN derives exactly from this observation capability. In turn, the ability to identify the most important convolutional layers represents the starting point for several tasks, one of which is the CNN compression of a CNN that we have seen in detail in this paper.

Our compression approach prunes entire layers; this reduces the number of parameters to be trained and speeds up both training and inference times. Interestingly, pruning a whole layer does not disrupt the nature of the CNN model itself, as it could happen with the cutting of redundant connections (see Section 2).

As for compression, we can draw further considerations by looking at which layers are typically pruned by our approach. With regard to this, we must preliminarily recall an important behavior typical of CNNs, i.e., the fact that the first convolutional layers extract high-level patterns from images (e.g., the shape of a dog), while the last ones focus on in-depth patterns (e.g., details of the dog, like its ears or its nose). From Figures 14 - 17, we can see that our compression approach does not generally cut the first convolutional layers, which are the ones close to the input; in fact, this cut is performed only when γ is high. One reason of this behavior concerns the fact that these layers probably extract generic patterns that are always useful for analyzing input images. This characteristic makes them essential, and so they are hardly cut off from our approach. Instead, the most pruned layers are the middle ones. This could depend on the excessive number of convolution operations applied on a fine feature map. Indeed, it could happen that the first 3-4 convolutional layers extract a meaningful pattern from images, which then undergoes other 7-8 convolutions and loses its significance. Finally, the last layers of VGG16, i.e., the ones close to the classification output, are cut off less than the middle layer but more than the initial ones. The reason for this behavior is not so obvious, and requires a further study in the future.

Although our multilayer network based representation has several interesting properties, it still has some limitations. For instance, it does not take into account the residual connections typical of ResNet architectures; therefore, ResNet cannot be represented through it. However, this kind of

connection can be easily added to our model, and we plan to make this task in the next future. Another limit concerns the view of the filters of a convolutional layer as a single unit. Indeed, our mapping approach aggregates all the computation results of the convolutional filters through a mean or a median. This aggregation keeps the multilayer network size low, but it loses information about single filters. Of course, this is a tradeoff between the computation time required to process a larger multilayer network, and the need for an in-depth analysis. However, this implies that we cannot prune any filter from the convolutional layers because we do not have the corresponding data within the multilayer network model. Actually, this issue can be addressed by making the multilayer network bigger, and then by developing a compression algorithm (similar to the one proposed here) to identify the most performing filters from the convolutional layers of a CNN.

Another limit concerns the datasets we employed here. MNIST and CALTECH-101 are surely two important benchmarks, and are heavily used by the research community. However, both of them are not as big as ImageNet, CALTECH-256, CIFAR100, which are much more complex and require much more computational power. We are confident that our approach can show good performances with all datasets, but it surely will be interesting to study its behavior on huge ones.

6 Conclusion

In this paper, we addressed the problem of representing a deep learning model in order to enable its next analysis, exploration and manipulation. In particular, we focused on a family of deep learning architectures, namely CNNs, and used a multilayer network, for their representation. Therefore, we proposed an approach to map each element of a CNN into the constructs of a multilayer networks, namely nodes, arcs, arc weights and layers. Then, in order to give an idea of the potential of the proposed representation approach, we used it in the context of a layer pruning method for a CNN. Finally, we presented an extensive experimental campaign aimed at showing the suitability of the proposed approach and at evaluating its performance.

Everything we have seen in this paper should not be considered as an endpoint but, on the contrary, as a starting point for further efforts in this research field. Indeed, at the end of the previous section, we have already seen some limitations of the approaches proposed in this paper. We have also seen how their overcoming provides the inspiration for one or more possible future developments of our research efforts.

References

[1] Dilek Altas, Ahmet M. Cilingirturk, and Vildan Gulpinar. Analyzing the process of the artificial neural networks by the help of the social network analysis. *New Knowledge Journal of Science*, 2:80–91, 2013.

[2] Jose M. Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 856–867, Red Hook, NY, USA, 2017. Curran Associates Inc.

[3] Ricardo F. Alvear-Sandoval, Jose L. Sancho-Gomez, and Anibal R. Figueiras-Vidal. On improving cnns performance: The case of mnist. *Information Fusion*, 52:106–109, 2019.

[4] Plamen Angelov and Eduardo Soares. Towards explainable deep neural networks (xdnn). *Neural Networks*, 130:185–194, 2020.

[5] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.*, 13(3), February 2017.

[6] Arash Ardakani, Carlo Condo, and Warren J. Gross. Sparsely-connected neural networks: Towards efficient VLSI implementation of deep neural networks. *CoRR*, abs/1611.01427, 2016.

[7] Mohammad Babaeizadeh, Paris Smaragdis, and Roy H. Campbell. A simple yet effective method to prune dense layers of neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[8] Federico Battiston, Vincenzo Nicosia, and Vito Latora. Structural measures for multiplex networks. *Phys. Rev. E*, 89:032804, Mar 2014.

[9] Shi Chen and Qi Zhao. Shallowing deep networks: Layer-wise pruning based on feature representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(12):3048–3056, 2019.

[10] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 2285–2294. JMLR.org, 2015.

[11] Yanjiao Chen, Baolin Zheng, Zihan Zhang, Qian Wang, Chao Shen, and Qian Zhang. Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions. *ACM Comput. Surv.*, 53(4), August 2020.

[12] Zhen Chen, Zhibo Chen, Jianxin Lin, Sen Liu, and Weiping Li. Deep neural network acceleration based on low-rank approximated channel pruning. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(4):1232–1244, 2020.

[13] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, pages 1–43, 2020.

[14] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 3123–3131, Cambridge, MA, USA, 2015. MIT Press.

[15] Shaveta Dargan, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar. A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering*, 27:1071–1092, 2020.

[16] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[17] Max Ferguson, Ronay Ak, Yung-Tsun Tina Lee, and Kincho H. Law. Automatic localization of casting defects with convolutional neural networks. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1726–1735, 2017.

[18] Nidhi Gowdra, Roopak Sinha, Stephen MacDonell, and Wei Qi Yan. Mitigating severe over-parameterization in deep convolutional neural networks through forced feature abstraction and compression with an entropy-based heuristic. *Pattern Recognition*, page 108057, 2021. Elsevier.

[19] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1387–1395, Red Hook, NY, USA, 2016. Curran Associates Inc.

[20] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, Las Vegas, Nevada, USA, 2016. IEEE.

[22] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, 2017.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[24] Fred Hohman, Haekyu Park, Caleb Robinson, and Duen Horng Polo Chau. Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1096–1106, 2020.

[25] Lu Hou and James T. Kwok. Loss-aware weight quantization of deep networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[26] Lu Hou, Quanming Yao, and James T. Kwok. Loss-aware binarization of deep networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[27] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[28] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[29] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4114–4122, Red Hook, NY, USA, 2016. Curran Associates Inc.

[30] Forrest N Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

[31] Yani Ioannou, Duncan P. Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[32] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[33] Minsuk Kahng, Pierre Y. Andrews, Aditya Kalro, and Duen Horng Chau. Activis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2018.

[34] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artif. Intell. Rev.*, 53(8):5455–5516, 2020.

[35] Jangho Kim, Seonguk Park, and Nojun Kwak. Paraphrasing complex network: Network compression via factor transfer. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[36] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[37] Mikko Kivela, Alex Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 07 2014.

[38] Xu Lan, Xiatian Zhu, and Shaogang Gong. Knowledge distillation by on-the-fly native ensemble. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[39] Chong Li and C.-J. Richard Shi. Constrained optimization based low-rank approximation of deep neural networks. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X*, volume 11214 of *Lecture Notes in Computer Science*, pages 746–761. Springer, 2018.

[40] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

[41] Jeng-Hau Lin, Tianwei Xing, Ritchie Zhao, Zhiru Zhang, Mani Srivastava, Zhuowen Tu, and Rajesh K. Gupta. Binarized convolutional neural networks with separable filters for efficient hardware acceleration. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 344–352, 2017.

[42] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014.

[43] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[44] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Penksy. Sparse convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 806–814, 2015.

[45] Jing Liu, Bohan Zhuang, Zhuangwei Zhuang, Yong Guo, Junzhou Huang, Jinhui Zhu, and Mingkui Tan. Discrimination-aware network pruning for deep model compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[46] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, 2017.

[47] Rushed Kanawati Manel Hmimida. Community detection in multiplex networks: A seed-centric approach. *Networks & Heterogeneous Media*, 10(1):71–85, 2015.

[48] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*, pages 6848–6856, Salt Lake City, Utah, USA, 2018. IEEE.

[49] Mohammed Amine Merzoug, Ahmed Mostefaoui, Mohammed H. Kechout, and Sebti Tamraoui. Deep learning for resource-limited devices. In *Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Q2SWinet '20, page 81–87, New York, NY, USA, 2020. Association for Computing Machinery.

[50] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[51] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[52] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[53] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6655–6659, 2013.

[54] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[56] Suraj Srinivas and R. Venkatesh Babu. Data-free parameter pruning for deep neural networks. *CoRR*, abs/1507.06149, 2015.

[57] Suraj Srinivas and Francois Fleuret. Knowledge transfer with Jacobian matching. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4723–4731. PMLR, 10–15 Jul 2018.

[58] Kenji Suzuki, Isao Horiba, and Noboru Sugie. A simple neural network pruning algorithm with application to filter synthesis. *Neural Process. Lett.*, 13(1):43–53, 2001.

[59] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[60] Zi-Rui Wang and Jun Du. Joint architecture and knowledge distillation in CNN for Chinese text recognition. *Pattern Recognition*, 111:107722, 2021. Elsevier.

[61] Zichao Yang, Marcin Moczulski, Misha Denil, Nando De Freitas, Le Song, and Ziyu Wang. Deep fried convnets. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1476–1483, 2015.

[62] Kaixuan Yao, Feilong Cao, Yee Leung, and Jiye Liang. Deep Neural Network Compression through Interpretability-Based Filter Pruning. *Pattern Recognition*, page 108056, 2021. Elsevier.

[63] Jiaxuan You, Jure Leskovec, Kaiming He, and Saining Xie. Graph structure of neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10881–10891. PMLR, 13–18 Jul 2020.

[64] Chun-Yang Zhang, Qi Zhao, C. L. Philip Chen, and Wenxi Liu. Deep compression of probabilistic graphical networks. *Pattern Recognition*, 96:106979, 2019. Elsevier.

[65] Quanshi Zhang, Ruiming Cao, Feng Shi, Ying Nian Wu, and Song-Chun Zhu. Interpreting cnn knowledge via an explanatory graph. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[66] Quanshi Zhang, Ruiming Cao, Ying N. Wu, and Song-Chun Zhu. Mining object parts from cnns via active question-answering. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3890–3899, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society.

[67] Quanshi Zhang, Yu Yang, Haotian Ma, and Ying Nian Wu. Interpreting cnns via decision trees. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6254–6263, 2019.

[68] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*, pages 6848–6856, Salt Lake City, Utah, USA, 2018. IEEE.

[69] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, 2016.

[70] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.

[71] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017.