

# Mapping and Compressing a Convolutional Neural Network through a Multilayer Network

(Discussion Paper)

Alessia Amelio<sup>1</sup>, Gianluca Bonifazi<sup>2</sup>, Enrico Corradini<sup>2</sup>, Michele Marchetti<sup>2</sup>,  
Domenico Ursino<sup>2</sup> and Luca Virgili<sup>2</sup>

<sup>1</sup>INGEO, University "G. D'Annunzio" of Chieti-Pescara

<sup>2</sup>DII, Polytechnic University of Marche

## Abstract

This paper falls in the context of the interpretability of the internal structure of deep learning architectures. In particular, we propose an approach to map a Convolutional Neural Network (CNN) into a multilayer network. Next, to show how such a mapping helps to better understand the CNN, we propose a technique for compressing it. This technique detects if there are convolutional layers that can be removed without reducing the performance too much and, if so, removes them. In this way, we obtain lighter and faster CNN models that can be easily employed in any scenario.

## Keywords

Deep Learning, Convolutional Neural Networks, Multilayer Networks, Convolutional Layer Pruning

## 1. Introduction

In recent years, we have witnessed a massive spread of deep learning models in many contexts [1, 2] with the goal of solving increasingly complex problems. The growing complexity of the problems to solve requires increasingly sophisticated models to achieve the best performance. However, more and more researchers realize the need to reduce the size and complexity of deep networks [3, 4]. As a result, enormous efforts are being made to introduce new architectures of deep learning networks that are more efficient and less complex. In parallel, several methods are being proposed to reduce the size of existing networks without affecting their performance too much [5, 6]. To this end, it is extremely important to be able to explore the various layers and components of a deep learning model. In fact, we could identify the most important components, the most interesting patterns and features, the information flow, and so on.

In this paper, we want to make a contribution in this setting. In particular, we start from the assumption that complex networks, and in particular multilayer ones, can significantly support the representation, analysis, exploration and manipulation of deep learning networks. Based on

---


SEBD 2022: The 30th Italian Symposium on Advanced Database Systems, June 19-22, 2022, Tirrenia (PI), Italy

✉ a.amelio@unich.it (A. Amelio); g.bonifazi@univpm.it (G. Bonifazi); e.corradini@pm.univpm.it (E. Corradini); m.marchetti@pm.univpm.it (M. Marchetti); d.ursino@univpm.it (D. Ursino); luca.virgili@univpm.it (L. Virgili)

🆔 0000-0002-3568-636X (A. Amelio); 0000-0002-1947-8667 (G. Bonifazi); 0000-0002-1140-4209 (E. Corradini); 0000-0003-3692-3600 (M. Marchetti); 0000-0003-1360-8499 (D. Ursino); 0000-0003-1509-783X (L. Virgili)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

this insight, we first propose an approach to map deep learning networks into multilayer ones and then use the latter to explore and manipulate the former.

We focus on one family of deep learning networks, namely Convolutional Neural Networks (hereafter, CNNs) [7]. Multilayer networks [8] are a type of complex networks sophisticated enough to represent all aspects of a CNN. In fact, through their fundamental components (i.e., nodes, arcs, weights and layers), they are able to represent all the typical concepts of a CNN (i.e., nodes, connections, filters, weights, etc.). Once we have mapped a CNN into a multilayer network, we can use the latter to study and manipulate the former. To give an idea of its potential, in the paper we will use it to support an approach to prune convolutional layers [9] from a CNN. This approach aims to identify if there are layers in the CNN that can be pruned without reducing the performance too much, and, if so, proceeds with pruning and returns a new CNN without those layers.

The outline of this paper is as follows: In Section 2, we describe our approach to mapping a CNN into a multilayer network. In Section 3, we describe the use of the multilayer network to prune one or more convolutional layers of the CNN. Finally, in Section 4, we draw our conclusion and take a look at some possible future developments of our research.

## 2. Mapping a Convolutional Neural Network into a multilayer network

In this section, we present our approach to mapping a CNN into a multilayer network. It represents the first contribution of this paper.

### 2.1. Class network definition

In this subsection, we describe a CNN by means of a single-layer network, referred to as a *class network*. It 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.

A CNN consists of  $M$  convolutional layers, each having  $x$  filters (also called “kernels”). In a convolutional layer, each filter slides over the input with a given stride and creates a feature map. The input  $\mathcal{I}$  of the first convolutional layer is given by the original image, while the input of the next convolutional layer is given by a feature map. Applying a filter on the element  $\mathcal{I}(i, j)$  of the input provides a new element  $\mathcal{O}(i, j)$  of the output feature map  $\mathcal{O}$  of the output. Based on this, the set  $V$  of the nodes of  $G$  consists of a set  $\{V_1, V_2, \dots, V_M\}$  of node subsets. The subset  $V_k$  indicates the contribution of the  $k^{th}$  convolutional layer  $c_k$ . This is obtained by applying the  $x_k$  filters of this layer to the input of  $c_k$ . Therefore, a node  $p \in V_k$  represents the output obtained by applying the  $x_k$  filters of  $c_k$  to some position  $(i, j)$  of the input.

Since by applying a filter on  $\mathcal{I}(i, j)$  we get a new element  $\mathcal{O}(i, j)$ , it is straightforward that there is a direct connection between  $\mathcal{I}(i, j)$  and  $\mathcal{O}(i, j)$ . Actually, there is a direct connection 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. As each convolutional layer  $c_k$  has a set of  $x_k$  filters, there are  $x_k$  sets of direct connections between  $\mathcal{I}(i, j)$  and  $\mathcal{O}_k(i, j)$ , one for each filter. Since the  $x_k$  filters are 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.

In a CNN, besides the convolutional layer there are pooling layers. A pooling layer shrinks the input feature map and leads to an increase in the number of connections between the input and the output. This is achieved by sliding the filter over the input with a given stride and determining an aggregate value (e.g., the maximum) for each filter window. As aggregate values are still elements of the feature map provided as input, the next application of a convolutional layer to the feature map returned by a pooling layer generates connections between the aggregate values and the elements of the feature map returned by the convolutional layer. In particular, there will be direct connections between the aggregate values of the feature map provided as input to the pooling layer and the adjacent elements of the feature map generated by the next convolutional layer.

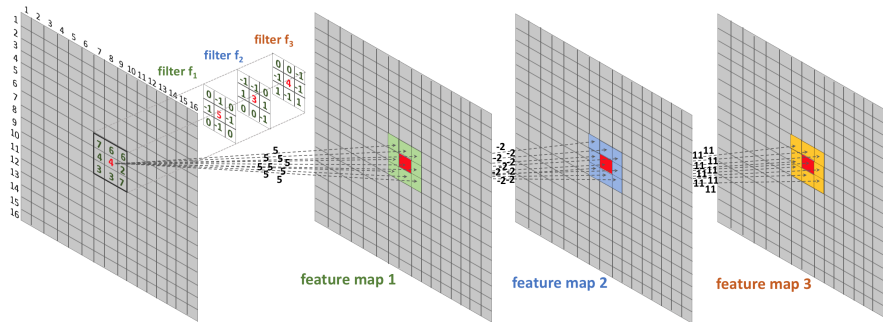
Therefore, based on the previous reasoning, we can say that the application of a filter to the element  $\mathcal{I}(i, j)$  generates a new element  $\mathcal{O}(i, j)$ , whose value is obtained by the following convolution operation:

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

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

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  $g(i, j)$ , representing the convolution result. In presence of  $x$  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  $g_1(i, j), g_2(i, j), \dots, g_x(i, j)$ .

Figure 1 illustrates the application of three filters of size  $3 \times 3$  (green, blue and yellow colored, respectively) to  $\mathcal{I}(8, 8)$ . For each filter  $f_h$ , it also shows 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 obtained as:  $g_1(8, 8) = f_1(8, 8) \cdot \mathcal{I}(8, 8) = 5$ ;  $g_2(8, 8) = f_2(8, 8) \cdot \mathcal{I}(8, 8) = -2$ ;  $g_3(8, 8) = f_3(8, 8) \cdot \mathcal{I}(8, 8) = 11$ .



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

To obtain the arcs of  $G$  from the weights of the direct connections of the  $x$  filters, we adopt some statistical descriptors. Our objective is to have only one set of arcs from the node

corresponding to  $\mathcal{I}(i, j)$  to the node corresponding to  $\mathcal{O}(i + s, j + t)$ ,  $-a \leq s \leq a$ ,  $-b \leq t \leq b$ <sup>1</sup>. The weight of an arc from  $\mathcal{I}(i, j)$  to  $\mathcal{O}(i + s, j + t)$  is obtained by applying a suitable statistical descriptors to the weights of  $g_h(i, j)$ ,  $1 \leq h \leq x$ . The two statistical descriptors we have chosen to adopt are the *mean* and the *median*, which achieved the best performance results.

As a consequence of the previous reasoning, the set  $E$  of the arcs of  $G$  consists of a set of subsets  $E = \{E_1, E_2, \dots, E_{M-1}\}$ . 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  $W = \{W_1, W_2, \dots, W_{M-1}\}$ . Here,  $W_k$  is the set of weights associated with the arcs of  $E_k$ .

## 2.2. Mapping a CNN into a multilayer network

After showing how it is possible to build a class network representing a CNN, in this section we illustrate how, with the support of the class network model and a dataset containing the training data, it is possible to build a multilayer network capable of fully mapping *both the CNN and its behavior*. Roughly speaking, a multilayer network is a set of  $t$  class networks, one for each target class in the dataset. Formally speaking, let  $D$  be a dataset consisting of  $t$  target classes  $Cl_1, Cl_2, \dots, Cl_t$ , and let  $cnn$  be a Convolutional Neural Network. The multilayer network  $\mathcal{G} = \{G^1, G^2, \dots, G^t\}$  corresponding to  $cnn$  is a set of  $t$  class networks such that  $G^h$ ,  $1 \leq h \leq t$ , corresponds to the  $h^{th}$  target class of  $D$ .

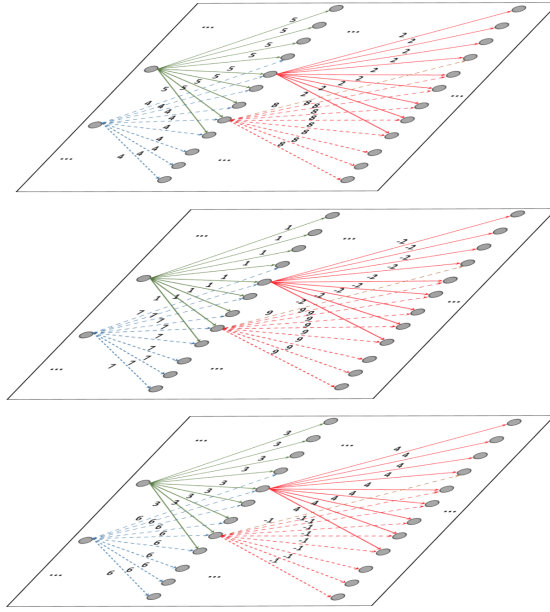
Figure 2 shows a multilayer network  $\mathcal{G}$  characterized by three layers,  $G^1$ ,  $G^2$  and  $G^3$  each consisting of a generated class network. Figure 3 shows the generation of a portion of  $\mathcal{G}$  obtained by extracting the three feature maps of three target classes from  $cnn$ . In this last network, a set of filters of size  $3 \times 3$  with stride 1, sliding over positions (3, 2) and (4, 2) of the feature maps, generates some arcs for the class networks of  $\mathcal{G}$ . The weights of the arcs are computed by applying the rules described in Section 2.1. In particular, 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. Similarly, a set of arcs of weights 2 (resp., -2, 4) and 8 (resp., 9, -1) are generated between the second and the third feature maps.

Our algorithm for constructing  $\mathcal{G}$  from  $cnn$  and  $D$  consists of two steps. During the first step, it creates a support data structure consisting of a *list of patch lists*. A patch list is a part of a feature map having the same size as the filters applied to the next convolutional layer and giving rise to a node in the multilayer network. The list of patch lists is then used during the second step to construct  $\mathcal{G}$ .

The function corresponding to the first step, called `CREATE_PATCHES`, receives  $cnn$  and a target class  $Cl_h$ ,  $1 \leq h \leq t$ . It operates on the feature maps provided in input to each convolutional layer of  $cnn$  and proceeds as follows. It uses a list `conv_layers` that initially contains the convolutional layers of  $cnn$ . Afterwards, it iterates over the elements of `conv_layers` and provides them with the images of  $Cl_h$ . During each iteration, it treats the current element as the *source* and the next element as the *target*. The feature map returned from *source* is given as input to *target*, which processes it as specified below.

At the beginning of each iteration, `CREATE_PATCHES` determines the starting and ending points of the source output. After that, it iterates over the source output and creates a patch for

<sup>1</sup>Here and in the following, we employ the symbols  $\mathcal{I}(i, j)$  and  $\mathcal{O}(i, j)$  for indicating both the elements of the feature maps and the corresponding nodes of the class network.



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

each application of the convolutional filter on an element. At the end of the iteration, it stores the patches corresponding to a convolutional layer in a list called *patch\_list*. Finally, it returns in output the lists corresponding to all convolutional layers of *cnn*.

The function corresponding to the second step, called `CREATE_LAYER_NETWORK` receives *cnn* and a target class  $Cl_h$ ,  $1 \leq h \leq t$  of the multilayer network  $\mathcal{G}$ . It first calls `CREATE_PATCHES` to receive the list of patch lists corresponding to *cnn* and  $Cl_h$ . Then, it creates an initially empty network  $G^h$ . Afterwards, it scrolls the list of patch lists returned by `CREATE_PATCHES` considering the current list as *source* and the next one as *target*.

For each iteration, it adds to  $G^h$  a node for each patch present in *source* or *target*, if they are not already present in  $G^h$ . Then, it determines the size of the area covered by the filter in *target*. This size is equal to the one covered in *source*, if we are in presence of a convolutional layer, or it is greater, if we have a pooling layer. After that, it iterates over the *source* and *target* nodes on which the filter acts. Given a node, it considers all source nodes that can be processed by the filter whose center falls in the rectangle defined by the coordinates of the patch of  $v_t$  and, for each of them, adds an arc from it to  $v_t$  in  $G^h$ .

Once this arc has been entered, it computes the corresponding weights by applying the formula seen above (see Figure 3 for an example of its application). At the end of its iterations, `CREATE_LAYER_NETWORK` has created the  $h^{th}$  layer  $G^h$  of  $\mathcal{G}$ , corresponding to the target class  $Cl^h$ . Applying this function  $t$  times, one for each target class in the dataset  $D$ , we obtain the final multilayer network.



As previously pointed out,  $\mathcal{G} = \{G^1, G^2, \dots, G^t\}$  has a layer for each target class. As a consequence, the overall degree  $\delta(v)$  of the node  $v$  in  $\mathcal{G}$  can be obtained by suitably aggregating the degrees  $d^1(v), d^2(v), \dots, d^t(v)$  of  $v$  in the  $t$  layers of  $\mathcal{G}$ . More specifically, if we indicate with  $\mathcal{F}$  a suitable aggregation function, we have that:

$$\delta(v) = \mathcal{F}(d^1(v), d^2(v), \dots, d^t(v)) \quad (1)$$

Let  $d^{tot}(v) = \sum_{h=1}^t d^h(v)$  be the sum of the degrees of  $v$  in the  $t$  layers of  $\mathcal{G}$ . We adopt the following entropy-based aggregation function [10, 11] for determining  $\delta(v)$ :

$$\delta(v) = - \sum_{h=1}^t \frac{d^h(v)}{d^{tot}(v)} \log \left( \frac{d^h(v)}{d^{tot}(v)} \right), \quad (2)$$

This function refers to the famous concept of information entropy introduced by Shannon [12]. It favors the presence of 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 and a low degree of  $v$  in many layers. In this way, we favor those nodes whose feature extraction is balanced for different target classes [13] and penalize those nodes that do not favor such a balance.

Our approach for the compression of *cnn* selects a subset of the nodes of  $\mathcal{G}$  with the highest values of  $\delta$ . This is equivalent to selecting the convolutional layers of *cnn* that contribute the most to the quality of the classification and, therefore, cannot be discarded in any way. In particular, our approach selects the nodes of  $\mathcal{G}$  whose values of  $\delta$  are higher than a certain threshold  $th_\delta = \gamma \cdot \bar{\delta}$ .

Here,  $\bar{\delta}$  denotes a statistical descriptor of the values of the overall degree  $\delta$  of all nodes in  $\mathcal{G}$ . Our approach allows the usage of the mean or the median as statistical aggregators because they are the ones that allowed us to obtain the best experimental results.  $\gamma$  is a scaling factor that allows us to tune the contribution of  $\bar{\delta}$ . Its value belongs to the real interval  $[0, +\infty)$ .

After selecting the subset of the nodes of  $\mathcal{G}$  with an overall degree  $\delta$  higher than  $th_\delta$ , our approach determines the set of the convolutional layers of *cnn* from which these nodes were extracted. Finally, it keeps these layers in the compressed version of *cnn* while discarding the others. At this point, our approach terminates by training the pruned network.

## 4. Conclusion

In this paper, we have seen how it is possible to map a deep learning architecture, specifically a CNN, into a multilayer network, which facilitates its analysis, exploration and manipulation. To this end, we proposed an approach to map each element of a CNN into the elements of the multilayer networks, namely nodes, arcs, arc weights and layers. Then, to give an idea of what can be done thanks to such a mapping, we proposed an approach to compress a CNN based on the pruning of the layers that least contribute to the quality of the classification.

The approach proposed in this paper is a starting point for further research efforts in this area. For example, our approach does not consider the residual connections, typical of ResNet. In the future, we plan to extend it in order to handle this type of architecture. Also, at the moment, our approach is not able to prune single filters from a convolutional layer. Doing so would require a

much more detailed mapping of the CNN into the multilayer network. At the moment, we have not done this to keep the size of the multilayer network limited. However, conceptually there are no limitations in performing such a task. In the future, we plan to proceed in that direction so that we can perform the pruning of a CNN with finer granularity.

## References

- [1] S. Dargan, M. Kumar, M. R. Ayyagari, G. Kumar, A survey of deep learning and its applications: A new paradigm to machine learning, *Archives of Computational Methods in Engineering* 27 (2020) 1071–1092.
- [2] E. Corradini, G. Porcino, A. Scopelliti, D. Ursino, L. Virgili, Fine-tuning SalGAN and PathGAN for extending saliency map and gaze path prediction from natural images to websites, *Expert Systems With Applications* 191 (2022) 116282. Elsevier.
- [3] Z. Chen, Z. Chen, J. Lin, S. Liu, W. Li, Deep neural network acceleration based on low-rank approximated channel pruning, *IEEE Transactions on Circuits and Systems I: Regular Papers* 67 (2020) 1232–1244. doi:10.1109/TCSI.2019.2958937.
- [4] J. Liu, B. Zhuang, Z. Zhuang, Y. Guo, J. Huang, J. Zhu, M. Tan, Discrimination-aware network pruning for deep model compression, *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021) 1–1. doi:10.1109/TPAMI.2021.3066410.
- [5] T. Choudhary, V. Mishra, A. Goswami, J. Sarangapani, A comprehensive survey on model compression and acceleration, *Artificial Intelligence Review* (2020) 1–43.
- [6] Z.-R. Wang, J. Du, Joint architecture and knowledge distillation in CNN for Chinese text recognition, *Pattern Recognition* 111 (2021) 107722. Elsevier.
- [7] A. Khan, A. Sohail, U. Zahoor, A. S. Qureshi, A survey of the recent architectures of deep convolutional neural networks, *Artif. Intell. Rev.* 53 (2020) 5455–5516. doi:10.1007/s10462-020-09825-6.
- [8] M. Kivela, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, M. A. Porter, Multilayer networks, *Journal of Complex Networks* 2 (2014) 203–271. doi:10.1093/comnet/cnu016.
- [9] S. Chen, Q. Zhao, Shallowing deep networks: Layer-wise pruning based on feature representations, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41 (2019) 3048–3056. doi:10.1109/TPAMI.2018.2874634.
- [10] R. K. Manel Hmimida, Community detection in multiplex networks: A seed-centric approach, *Networks & Heterogeneous Media* 10 (2015) 71–85.
- [11] F. Battiston, V. Nicosia, V. Latora, Structural measures for multiplex networks, *Phys. Rev. E* 89 (2014) 032804. doi:10.1103/PhysRevE.89.032804.
- [12] C. E. Shannon, A mathematical theory of communication, *The Bell System Technical Journal* 27 (1948) 379–423. doi:10.1002/j.1538-7305.1948.tb01338.x.
- [13] N. Gowdra, R. Sinha, S. MacDonell, W. Q. Yan, Mitigating severe over-parameterization in deep convolutional neural networks through forced feature abstraction and compression with an entropy-based heuristic, *Pattern Recognition* (2021) 108057. Elsevier.