Representation and Compression of Residual Neural Networks through a Multilayer Network Based Approach

Representation and Compression of Residual Neural Networks through a Multilayer Network Based Approach

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

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

Abstract

In recent years different types of Residual Neural Networks (ResNets, for short) have been introduced to improve the performance of deep Convolutional Neural Networks. To cope with the possible redundancy of the layer structure of ResNets and to use them on devices with limited computational capabilities, several tools for exploring and compressing such networks have been proposed. In this paper, we provide a contribution in this setting. In particular, we propose an approach for the representation and compression of a ResNet based on the use of a multilayer network. This is a structure sufficiently powerful to represent and manipulate a ResNet, as well as other families of deep neural networks. Our compression approach uses a multilayer network to represent a ResNet and to identify the possible redundant convolutional layers belonging to it. Once such layers are identified, it prunes them and some related ones obtaining a new compressed ResNet. Experimental results demonstrate the suitability and effectiveness of the proposed approach.

Keywords: Residual Neural Networks; Convolutional Neural Networks; Complex Networks; Multilayer Networks; Compression algorithm; Convolutional Layer Pruning

1 Introduction

Residual Neural Networks (hereafter, ResNets) were introduced as an advancement of traditional Convolutional Neural Networks (hereafter, CNNs) in pattern recognition (K. He, Zhang, Ren, & Sun, 2016a; Dobbs & Ras, 2022). ResNets make it possible to overcome the limitations of adding multiple layers to CNNs to solve complex problems. In fact, such addition results in lowering the performance of CNNs due to the optimization function, the network initialization and the vanishing gradient problem. To overcome these limitations, ResNets introduce residual blocks, characterized by skip connections, which link non-consecutive layers in the network. Skip connections are intended to solve the vanishing gradient problem and improve network performance. In this way, the gradient can flow through these shortcut connections. Skip connections also allow additional layers in the model to learn the identity function, which equals their outputs to their inputs to avoid the degradation of model performance. As a result of this, ResNets show equal or better performance than classical CNNs, as well as significantly better performance than neural networks with more layers.

Due to their ability to solve complex tasks in pattern recognition, ResNets have been employed extensively in various settings (Zhou et al., 2021; Kim, Jung, Park, Lee, & Ahn, 2022; Liu et al., 2021). These include:

- the medical context, where they have been adopted in multiple tasks, e.g., to distinguish COVID-19 cases from other pneumonia cases (Farooq & Hafeez, 2020), to diagnose cardiomegaly (Yoo, Han, & Chung, 2021), and to detect and classify brain tumors (Kumar, Prasad, & Metan, 2022);
- fault detection and diagnosis, in terms of anomaly detection (Amini & Zhu, 2022), prediction of remaining useful life for machineries (Zeng, Li, Jiang, & Song, 2021), fault detection on seismic structural images (Gao, Huang, & Zheng, 2021), etc.;
- defect detection, in terms of printed circuit board cosmetic defect detection (H. Zhang, Jiang, & Li, 2021) and automatic detection of steel surface defects (Damacharla, Rao, Ringenberg, & Javaid, 2021);

- human action and activity recognition (Ronald, Poulose, & Han, 2021; Kang, Zhang, Li, & Zhuo, 2021).

In order to perform the various tasks mentioned above, ResNets with a different number of layers have been adopted. Among them, we cite ResNets with 18 and 20 layers, ResNets with 50 and 56 layers, and, finally, ResNets with 110 and 152 layers.

Several variants and extensions of ResNets have been proposed in the literature in recent years (see, for instance, (Huang, Liu, Maaten, & Weinberger, 2017; Xie, Girshick, Dollár, Tu, & He, 2017)). One of them regards an enhanced version of ResNet (also called Version 2) (K. He, Zhang, Ren, & Sun, 2016b), which refines the residual block with a pre-activation feature allowing gradients to flow through shortcut connections to any of the previous layers.

Other papers (for instance, (Huang, Sun, Liu, Sedra, & Weinberger, 2016; Veit, Wilber, & Belongie, 2016)) showed that some of the layers of a ResNet could be redundant. ResNet compression may satisfy requirements related to saving energy, space and time in training and using these networks. It may also meet the need to employ these architectures on devices with reduced computational resources. Recently, several methods for automatically compressing deep neural networks have been proposed in the literature (Choudhary, Mishra, Goswami, & Sarangapani, 2020). Some of them, belonging to the categories of pruning (Abbasi-Asl & Yu, 2017; Chen & Zhao, 2019; Y. He, Zhang, & Sun, 2017; H. Li, Kadav, Durdanovic, Samet, & Graf, 2016; Luo, Wu, & Lin, 2017; Luo & Wu, 2017; Ma et al., 2019), low-rank factorization (Wang, Sun, Eriksson, Wang, & Aggarwal, 2018; Minnehan & Savakis, 2019), and knowledge distillation (Koratana, Kang, Bailis, & Zaharia, 2019; Aghli & Ribeiro, 2021), have been tested on different types of ResNets obtaining competitive results in terms of compression rate and classification accuracy.

In this paper, we want to provide a contribution in this setting. In particular, we propose some possible advances in the direction of deep neural network compression with special reference to ResNets. To this end, we propose a new approach to represent, explore, manipulate and compress ResNets. Our approach is based on complex networks, in particular on multilayer networks (Kivelä et al., 2014). A multilayer network is an extension of the concept of graph that, in addition to nodes and arcs, includes the concept of layer. A classical graph can be considered as a single-layer network. Therefore, a multilayer network consists of two or more layers, each of which consists of a graph. We will focus on ResNet Version 2 (K. He et al., 2016b) although our approach is general enough that it can be easily extended to other types of deep neural networks.

Our approach first transforms a ResNet into a multilayer network, which is powerful enough to model the ResNet but is much easier to represent and manipulate than the latter. The resulting multilayer network can capture all the main features that characterize a ResNet. In fact, the mapping from a ResNet to a multilayer network realized by our approach allows each aspect of the former, namely layers, filters and skip connections, to be mapped in terms of nodes, arcs, arc weights and layers of the latter.

The multilayer network thus obtained can be analyzed and manipulated using the typical concepts of complex network theory. Each analysis and manipulation of it corresponds to an analysis and manipulation of the ResNet represented by it. In order to give an idea of the potential of the multilayer network thus obtained, in this paper we show how it can be used to compress the corresponding ResNet

through the pruning of convolutional layers (Chen & Zhao, 2019). The purpose of the compression task is to identify convolutional layers that can be easily pruned from the network without losing too much information, thus obtaining a reduced version of the original ResNet. Regarding this, we point out that, thanks to the support of the multilayer network, our ResNet compression approach could also help the explanation of ResNets. In fact, it is able to "shed a light" on the black box underlying the ResNet by providing a justification as to why some layers, rather than others, are chosen to be pruned.

Finally, we want to point out that the approach to compress a ResNet proposed in this paper represents only one of the possible applications of the idea of modeling a ResNet through a multilayer network. In fact, as we will see below, the latter framework is capable of modeling a ResNet while preserving all its main features, such as convolutions and skip connections. At the same time, it allows the application of concepts, parameters and approaches typical of (Social) Network Analysis to the context of ResNets, which can foster to address in a new way several challenging issues concerning ResNets. Thanks to this, it is possible to perform a variety of investigations and applications on ResNets, with the advantage of not having to treat them as black boxes. In fact, as mentioned above, the capability of mapping all of the constructs of ResNets while preserving all their characteristics, allows us to provide explanations of the various choices made and to illuminate, at least partially, the black boxes underlying them.

The outline of this paper is as follows: Section 2 describes the related literature. Section 3 presents the part of our approach devoted to mapping a ResNet into a multilayer network. Section 4 describes our compression approach based on layer pruning. Section 5 illustrates the experiments we conducted to test our approach. Finally, Section 6 draws our conclusion and outlines some possible future developments of this research.

2 Related Literature

Deep CNNs are increasingly being used to perform more and more complex tasks that require high performance in short amounts of time. These tasks involve computer vision and are often carried out on devices with limited computational power. This has prompted many authors to focus on defining approaches to reduce the complexity of the networks and the time required to achieve results (Choudhary et al., 2020; Kim, Khan, & Kyung, 2019; Ghimire, Kil, & Kim, 2022). Such approaches have been proposed for various types of deep CNNs, including ResNets. Indeed, such networks can be very deep and, as a result, have thousands of parameters, which increases the time required for training and inference.

The techniques most commonly used to reduce the complexity of a network and the number of its parameters generally evaluate the usefulness of each convolutional filter and remove those providing less information. Using this procedure, the authors of (Abbasi-Asl & Yu, 2017) introduce CAR (Classification Accuracy Reduction), a structural compression scheme that can reduce the size of CNNs while minimizing the loss of accuracy compared to the original model. In (Y. He et al., 2017), the authors propose an algorithm that can accelerate already trained neural networks through a channel pruning algorithm applied on each layer, which aims to drop redundant channels. In (Luo & Wu, 2017), the authors propose a method to accelerate and compress CNNs. The value of each filter

is initially obtained through an entropy-based approach. This first computes the value of entropy relative to each filter. Then, it prunes the least important filters. After that, it performs a fine-tuning step to increase the generalization capability of the network that had been decreased by the previous pruning operation. The authors of this approach show that it is able to accelerate and compress ResNets while achieving only a slight decrease in accuracy.

Other similar methods are described in (Wang et al., 2018) and (Luo et al., 2017). The former introduces Tensor Ring Networks (TR-Nets), an approach capable of compressing both fully connected and convolutional layers of a neural network. The latter proposes a different criterion for pruning filters based on metrics obtained from the next layer, instead of the current one. In this way, the usefulness of a filter is determined based on the quality of information it passes to the next layer. These two approaches achieve similar results in terms of size reduction and accuracy preservation. Two other papers introducing approaches to compress ResNets using pruning techniques are (Chen & Zhao, 2019) and (H. Li et al., 2016). In particular, the authors of (Chen & Zhao, 2019) propose a layer-wise parameter pruning approach that can reduce the complexity of a neural network by eliminating redundant parameters and removing less important layers. In this way, they are able to significantly reduce the computational cost without sacrificing performance. Instead, the authors of (H. Li et al., 2016) propose an acceleration approach for CNNs that removes the filters having less effect on the output of the network, thus speeding up the whole learning process. The authors of (Ma et al., 2019) present a framework for deep neural network reduction that performs well when applied to a ResNet. This framework uses the Alternating Direction Method of Multipliers (ADMM) (Boyd, Parikh, Chu, Peleato, & Eckstein, 2011) to increase its performance by executing different types of pruning.

In the literature, approaches for compressing deep neural networks based on very different principles from those seen so far have also been proposed. For example, the authors of (Minnehan & Savakis, 2019) present an approach, called Cascaded Projection (CaP), based on projection techniques. This approach initially projects the filter channels of successive layers onto a unified low dimensional space; for this purpose, it uses low-rank projection. Then, in order to reduce classification loss, it minimizes the difference between the features of the next layer in the original model and the compressed one. For this purpose, it uses the Stochastic Gradient Descent (SGD) (Bottou, 2012) algorithm. In (Y. He et al., 2018), the authors propose AutoML for Model Compression (AMC), a compression method based on reinforcement learning. The authors also propose two schemes to perform resource-constrained compression and accuracy-guaranteed compression. In this way, they achieve higher mean average precision than the best hand-crafted pruning methods under the same compression ratio.

Finally, the authors of (Koratana et al., 2019) propose Learned Intermediate representation Training (LIT), a model that uses a deep neural network (called teacher) to train another neural network (called student). The advantage of this approach is that the latter network is smaller in size than the former. LIT compares the intermediate representations of the teacher and student networks before providing the student with the intermediate representation of the teacher in the previous block. Through this way of proceeding, it is able to transfer knowledge from a more extensive CNN to a smaller one, having fewer parameters, which can be used on devices with lower computational resources.

3 Mapping a ResNet into a multilayer network

In this section, we introduce our method for mapping a ResNet into a multilayer network. It represents the first part of our approach, as well as the first contribution of this paper. Using this method, it is possible to represent, analyze, manipulate and compress a ResNet through a multilayer network.

In Figure 1, we provide a graphic schematization of our approach.

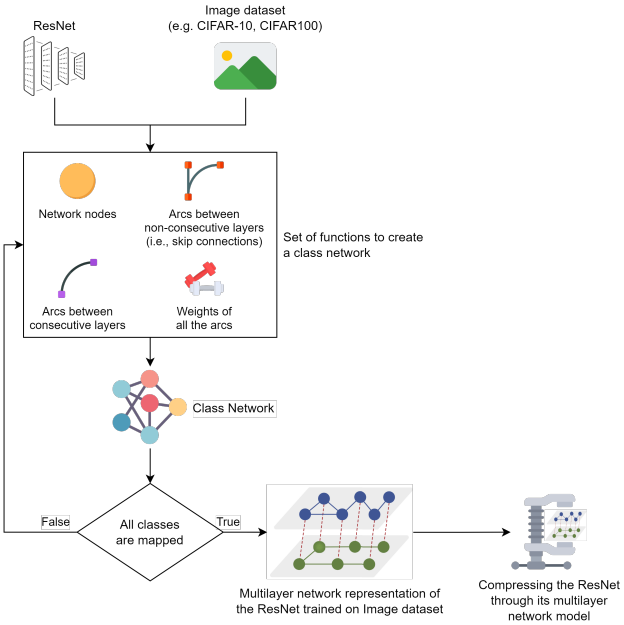
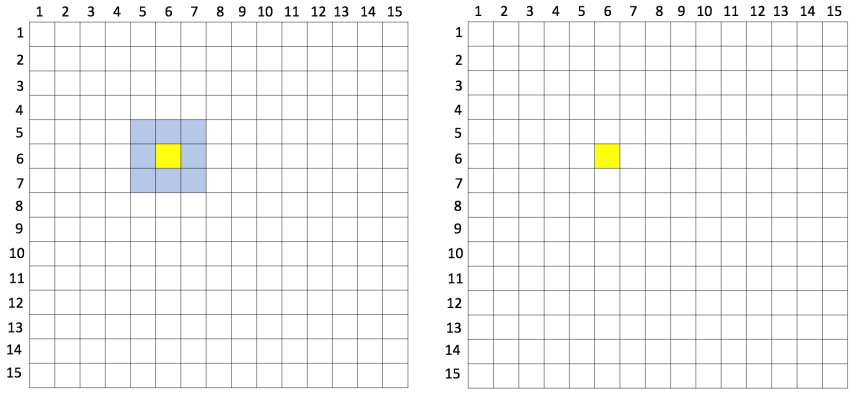


Figure 1: Overview of our approach to map a ResNet into a multilayer network and to compress the former through the latter

In this figure, we can see that our approach receives two inputs, namely a ResNet and an image dataset on which it was trained. For each class in the image dataset, it creates a graph representing a layer of a multilayer network (Kivelä et al., 2014). Specifically, the nodes of the graph represent the convolution operations performed by the ResNet on a specific image location, while the arcs denote the connections between consecutive or non-consecutive ResNet layers. The weights of the arcs are calculated based on the feature maps of the ResNet (see Section 3.2 for all details). The graph associated with a given layer of the multilayer network is called "class network" and represents the behavior of the ResNet on a specific class of images. The final step of our mapping approach involves creating the suitable links between all class networks (each of which consists of a graph and forms a layer) to form the multilayer network. This new ResNet representation allows us to perform many different tasks. For example, in this paper, to give an idea of a possible task, we illustrate how a ResNet can be compressed through the corresponding multilayer network (see Section 4).

This section is organized as follows: First, we provide the formal description of a ResNet, which we adopt in this paper (Section 3.1). Then, we introduce a new data structure called *class network*, which represents the foundation of our mapping method (Section 3.2). Finally, we provide an initially intuitive and then formal presentation of the latter (Section 3.3).



(a) (b)

Figure 2: (a) Filter of size 3×3 applied to $\mathcal{I}(6,6)$; (b) creation of the new element $\mathcal{O}(6,6)$

3.1 A ResNet formalization

A ResNet is made up of L convolutional layers, each consisting of several filters, called "kernels". A convolutional layer produces a feature map through the application of its filters that slide over the input with a given stride. The input $\mathcal{I}$ can be the original image, if the convolutional layer is the first in the ResNet, or a feature map, in the other cases. Applying a filter to the element (i,j) of the input $\mathcal{I}$ (hereafter $\mathcal{I}(i,j)$) returns a new element $\mathcal{O}(i,j)$ of the output feature map $\mathcal{O}$. For example, in Figure 2(a), a filter of size 3×3 is applied to the element $\mathcal{I}(6,6)$, resulting in the new element $\mathcal{O}(6,6)$ in Figure 2(b). In Figure 2(a), the area in which the filter acts is colored in blue and yellow, while, in Figure 2(b), the new element is colored in yellow.

In case of skip connection, there are two inputs $\mathcal{I}_1$ and $\mathcal{I}_2$. Applying a filter to the elements $\mathcal{I}_1(i,j)$ and $\mathcal{I}_2(i,j)$ returns a new element $\mathcal{O}(i,j)$ of the output feature map $\mathcal{O}$, obtained by suitably combining the results of the application of $\mathcal{I}_1(i,j)$ and $\mathcal{I}_2(i,j)$ to $\mathcal{O}(i,j)$. The operator used for such a combination is the sum.

The generation of a new element $\mathcal{O}(i,j)$ through the application of a filter to the element $\mathcal{I}(i,j)$ results in a direct connection between $\mathcal{I}(i,j)$ and $\mathcal{O}(i,j)$, as well as a direct connection between $\mathcal{I}(i,j)$ and each element adjacent to $\mathcal{O}(i,j)$ within the filter area. These latter connections allow the context information to be tracked. For instance, Figure 3 shows the direct connections between $\mathcal{I}(6,6)$, to which we apply a filter of size 3×3, and the elements $\mathcal{O}(6+u, 6+v)$, $-1 \leq u \leq 1$ and $-1 \leq v \leq 1$.

Since a convolutional layer c_l has a set of x_l filters, there are x_l sets of direct connections, like those in Figure 3, between $\mathcal{I}(i,j)$ and $\mathcal{O}(i,j)$, one for each filter. The application of each filter gives rise to a different value of $\mathcal{O}(i,j)$. Therefore, we have x_l values of $\mathcal{O}(i,j)$, i.e., $\mathcal{O}_1(i,j)$, $\mathcal{O}_2(i,j)$, $\cdots$, $\mathcal{O}_{x_l}(i,j)$, one for each filter. For example, in Figure 4, three different filters of size 3×3 are applied to the element $\mathcal{I}(6,6)$. This application returns three output elements $\mathcal{O}_1(6,6)$, $\mathcal{O}_2(6,6)$ and $\mathcal{O}_3(6,6)$. There is a direct connection between the element $\mathcal{I}(i,j)$ and the elements at the positions $\mathcal{O}_h(6+u, 6+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$, $1 \leq h \leq 3$.

By applying the x_l filters to the different input positions with a given stride, we obtain a set of similar connections to the feature maps for each position of the input.

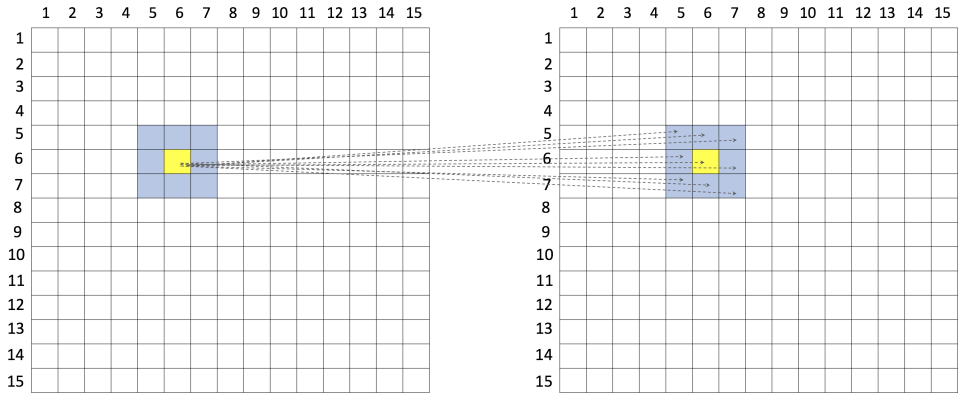


Figure 3: Dashed lines representing direct connections between $\mathcal{I}(6,6)$ and the elements $\mathcal{O}(6+u,6+v)$, $-1 \leq u \leq 1$ and $-1 \leq v \leq 1$ (yellow and blue colored)

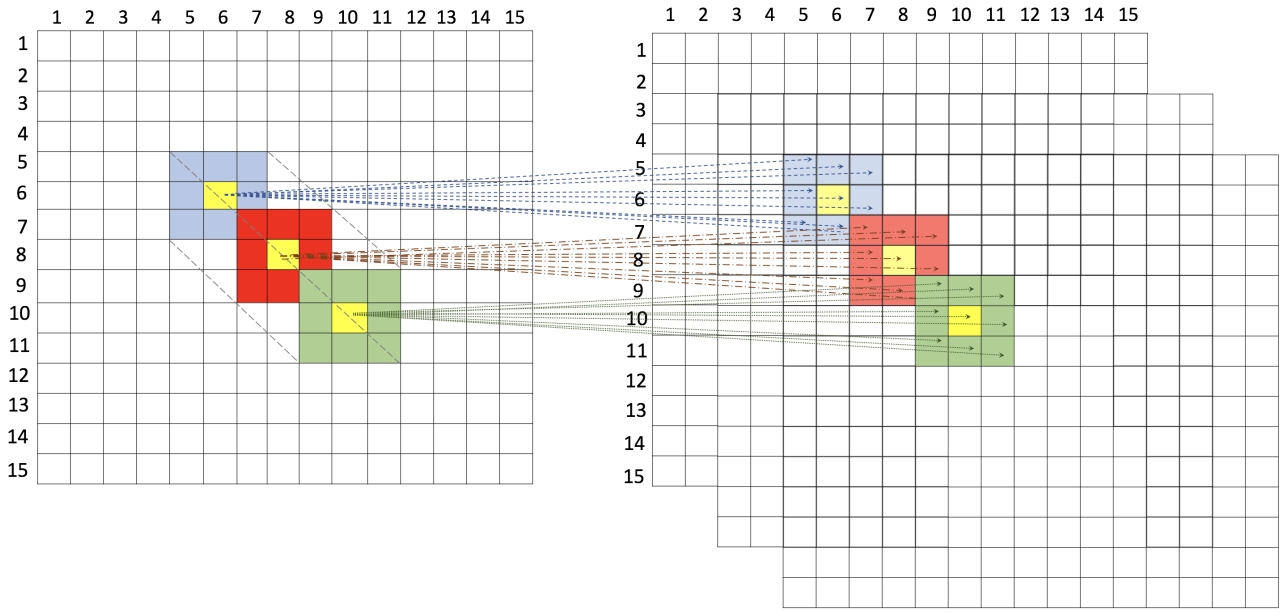


Figure 4: Dashed lines representing direct connections between $\mathcal{I}(6,6)$ and $\mathcal{O}_h(6+u,6+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$, $1 \leq h \leq 3$, obtained from the application of three filters (blue, red and green colored)

In case of skip connections, applying a filter to $\mathcal{I}_1(i,j)$ and $\mathcal{I}_2(i,j)$ results in direct connections, like those in Figure 3, between $\mathcal{I}_1(i,j)$ and $\mathcal{O}(i,j)$ and its adjacent elements, and between $\mathcal{I}_2(i,j)$ and $\mathcal{O}(i,j)$ and its adjacent elements. Since a convolutional layer c_l has x_l filters, there will actually be x_l sets of direct connections, like those in Figure 4, between $\mathcal{I}_1(i,j)$ and $\mathcal{O}_h(i,j)$ and its adjacent elements, and between $\mathcal{I}_2(i,j)$ and $\mathcal{O}_h(i,j)$ and its adjacent elements, $1 \leq h \leq x_l$.

As for the pooling layer, it reduces the size of the input feature map, lowering the number of connections between the input and output. This lowering is achieved by sliding the filter over the

input with a given stride and computing the maximum value on each filter window. Given that the maximum values are elements of the input feature map, when a convolutional layer is applied to the feature map returned by a pooling layer, direct connections are generated between the maximum values and the elements of the feature map returned by the convolutional layer. Specifically, there are direct connections from the maximum 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.

Figure 5 shows the generation of direct connections for a pooling layer. On the left, a pooling filter of size 2×2 and stride 2 is applied to the input. In the feature map, the colored elements represent the maximum values. On the right, the next application of a convolutional filter of size 3×3 to the position $\mathcal{I}(2,2)$ generates 9 direct connections between $\mathcal{I}(2,2)$ and $\mathcal{O}(2+u, 2+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$.

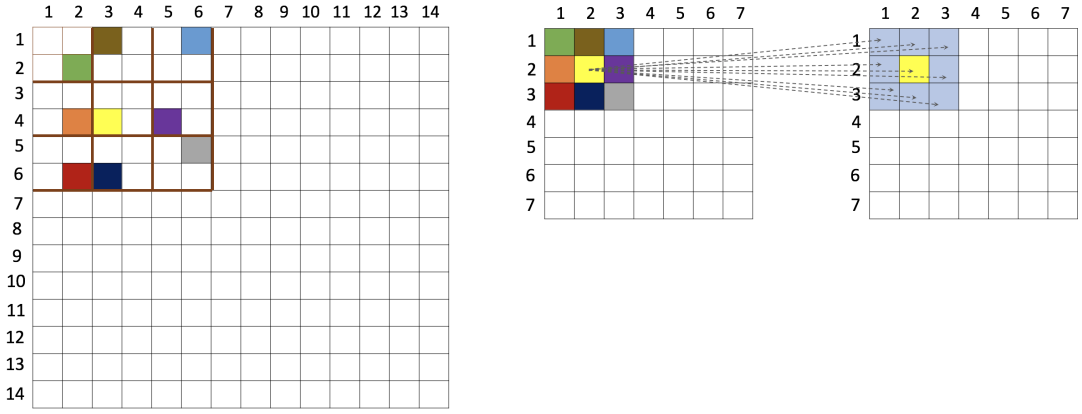


Figure 5: Generation of direct connections for a pooling layer with a filter of size 2×2 and stride 2. For each filter application, a colored element corresponds to the maximum value. At right, dashed lines represent the 9 direct connections between the maximum value at position $\mathcal{I}(2,2)$ and the elements $\mathcal{O}(2+u, 2+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$

Applying a filter to the element $\mathcal{I}(i,j)$ produces the output $\mathcal{O}(i,j)$, whose value is given by the following convolution operation:

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

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

A weight $A(g(i,j))$, corresponding to the result of the application of the activation function $A()$ to the output $g(i,j)$ of the convolution operation, is associated with the direct connections generated between $\mathcal{I}(i,j)$ and $\mathcal{O}(i+u, j+v)$, $-a \leq u \leq a$, $-b \leq v \leq b$.

In the rest of this section, in order not to burden the examples, we will use the identity activation function, whose output is identical to the input, and thus to the convolution result, i.e., $A(g(i,j)) = g(i,j)$.

In Figure 6, a filter of size 3×3 is applied to $\mathcal{I}(6,6)$; it returns the values of $\mathcal{O}(6+u, 6+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$. The weight corresponding to the convolution result is given by $g(6,6) = f(6,6) * \mathcal{I}(6,6) = (-1 \cdot 4) + (-1 \cdot 4) + (0 \cdot 5) + (0 \cdot 3) + (4 \cdot 3) + (0 \cdot 5) + (-1 \cdot 3) + (-1 \cdot 3) + (-1 \cdot 2) = -4$.

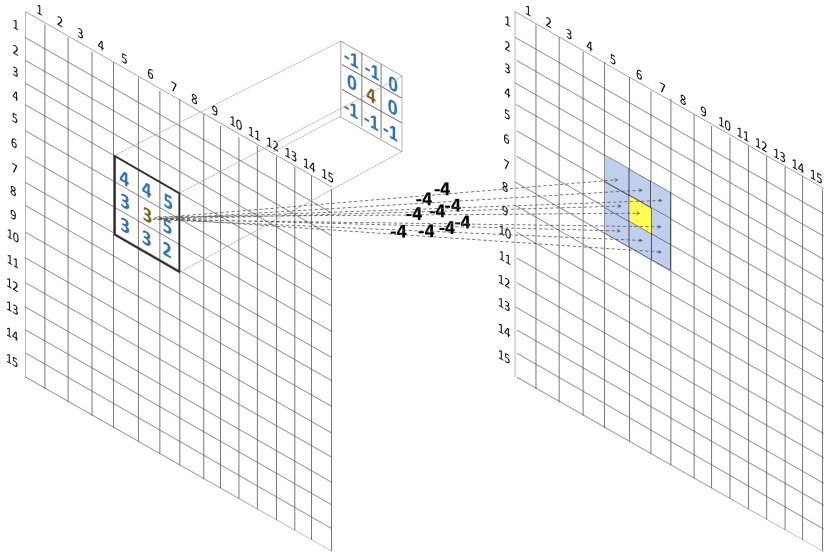


Figure 6: Filter of size 3×3 applied to $\mathcal{I}(6,6)$ (yellow colored) and computation of the weights for the direct connections between $\mathcal{I}(6,6)$ and $\mathcal{O}(6+u, 6+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$

Given a number x_l of filters, the weights of the direct connections between $\mathcal{I}(i,j)$ and $\mathcal{O}_h(i+u, j+v)$, $-a \leq u \leq a$, $-b \leq v \leq b$, $1 \leq h \leq x_l$, are given by the values obtained by applying the activation function $A()$ to the outputs of the corresponding convolution operation, i.e., $A(g_1(i,j)), A(g_2(i,j)), \ldots, A(g_{x_l}(i,j))$. In Figure 7, three filters of size 3×3 (blue, red and green colored, respectively) are applied to $\mathcal{I}(6,6)$. This figure also shows, for each filter f_h, $1 \leq h \leq 3$, the weighted direct connections generated between $\mathcal{I}(6,6)$ and $\mathcal{O}_h(6+u, 6+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$. The three weights derive from the following convolution results: $g_1(6,6) = f_1(6,6) * \mathcal{I}(6,6) = -4$; $g_2(6,6) = f_2(6,6) * \mathcal{I}(6,6) = 8$; $g_3(6,6) = f_3(6,6) * \mathcal{I}(6,6) = 13$.

From the weights of the direct connections of the x_l filters linking the element $\mathcal{I}(i,j)$ to the element $\mathcal{O}_h(i,j)$, $1 \leq h \leq x_l$, some statistical descriptors can be computed for the weights of $A(g_h(i,j))$. In particular, we will adopt two statistical descriptors, namely the mean and the median, which have been shown to be the ones capable of guaranteeing the best performance.

The mean is defined as:

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

The median is defined as:

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

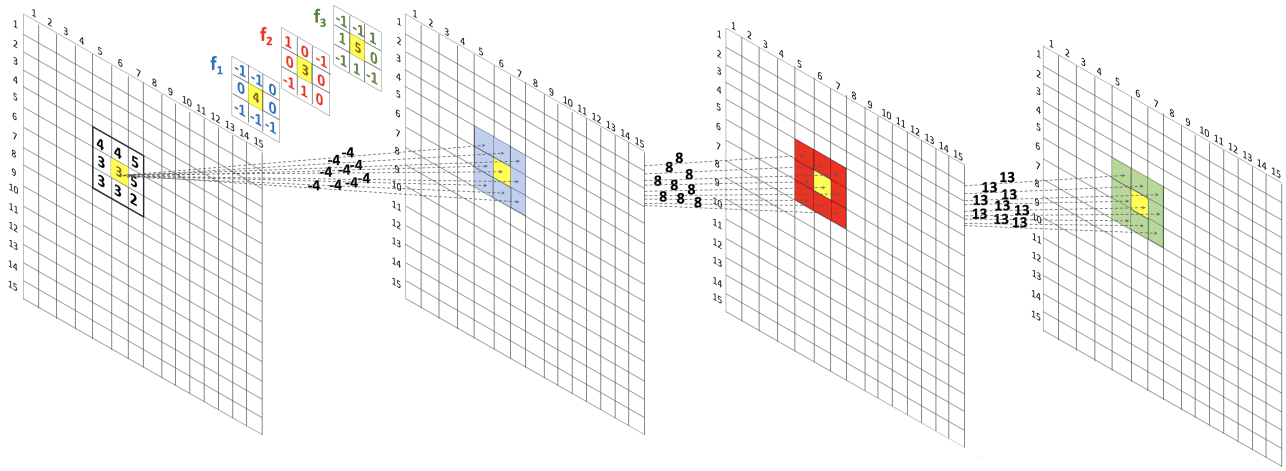


Figure 7: Three filters of size 3×3 (blue, red and green colored, respectively) applied to $\mathcal{I}(6,6)$. Computation, for each filter, of the weights for the direct connections between $\mathcal{I}(6,6)$ and $\mathcal{O}_h(6+u, 6+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$, $1 \leq h \leq 3$

These statistical descriptors are used to define the weight of the direct connections between the element $\mathcal{I}(i,j)$ and the elements $\mathcal{O}(i+u, j+v)$, $-a \leq u \leq a$, $-b \leq v \leq b$.

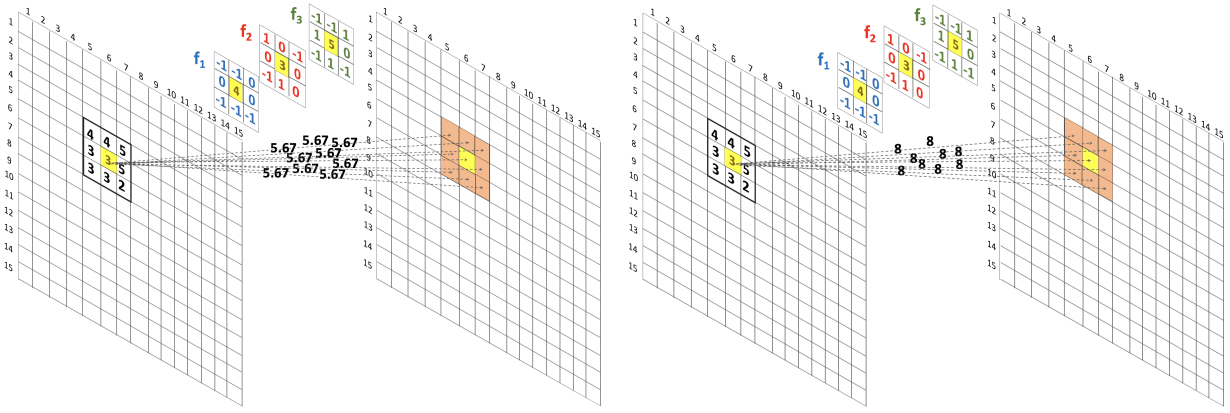
In Figure 8, the direct connections between $\mathcal{I}(6,6)$ and $\mathcal{O}(6+u, 6+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$ are weighted with the mean value (on the left) and the median one (on the right) of the connections related to the three filters shown in Figure 7. Specifically, the mean value is computed as $\frac{-4+8+13}{3} = 5.67$, while the median one is equal to 8.

As for skip connections, we have seen that applying a filter to $\mathcal{I}_1(i,j)$ and $\mathcal{I}_2(i,j)$ generates direct connections between $\mathcal{I}_1(i,j)$ and $\mathcal{O}(i,j)$ and between $\mathcal{I}_2(i,j)$ and $\mathcal{O}(i,j)$. The weights of these connections correspond to the results of the application of the activation function $A()$ to the outputs $g^1(i,j) = f(i,j) * \mathcal{I}_1(i,j)$ and $g^2(i,j) = f(i,j) * \mathcal{I}_2(i,j)$ of the convolution operations. More specifically, a weight $A(g^1(i,j))$ (resp., $A(g^2(i,j))$) is associated with the direct connections between $\mathcal{I}_1(i,j)$ (resp. $\mathcal{I}_2(i,j)$) and $\mathcal{O}(i+u, j+v)$, $-a \leq u \leq a$, $-b \leq v \leq b$. As in the case of simple connection between the feature maps, for the arc from $\mathcal{I}_1(i,j)$ (resp. $\mathcal{I}_2(i,j)$) to $\mathcal{O}(i+u, j+v)$, the weight is computed by calculating the mean or median of the weights of $A(g_h^1(i,j))$ (resp. $A(g_h^2(i,j))$) , $1 \leq h \leq x_l$.

3.2 Class network definition

Having provided our formalization of the ResNet, we are now able to introduce the class network, which plays a key role in the next mapping task. A *class network* is a single-layer network consisting of a directed weighted graph $G = (N, A, W)$, where N is the set of nodes, A is the set of arcs, and W is the set of arc weights.

N consists of a set $\{N_1, N_2, ..., N_L\}$ of node subsets, where L is the overall number of convolutional layers of the ResNet. A node $n \in N_l$, $1 \leq l \leq L$, corresponds to an element of $\mathcal{O}(i,j)$ and is derived by applying the l^{th} convolutional layer c_l to an element $\mathcal{I}(i,j)$ of the $(l-1)^{th}$ convolutional layer c_{l-1}. In turn, this implies the application of the x_l filters of c_l to $\mathcal{I}(i,j)$. In case of skip connection



(a) mean (b) median

Figure 8: Weights of the arcs from $\mathcal{I}(6,6)$ to $\mathcal{O}(6+u,6+v)$, $-1 \leq u \leq 1$, $-1 \leq v \leq 1$, achieved by adopting the mean (on the left) and the median (on the right) as statistical descriptors of the corresponding direct connections

of length $\lambda > 0$, n is derived by applying both the l^{th} and the $(l-\lambda)^{th}$ convolutional layers to $\mathcal{I}(i,j)$ and combining the corresponding outputs as shown in Section 3.1. Again, applying c_l (resp., $c_{l-\lambda}$) to $\mathcal{I}(i,j)$ implies the application of its x_l (resp., $x_{l-\lambda}$) filters to $\mathcal{I}(i,j)$.

A is the set of arcs of G. It can be represented as a set of subsets $A = \{A_1, A_2, \cdots, A_{L-1}\}$. There is an arc in A_l, $1 \leq l \leq L-1$ for each direct connection between a node of N_l and a node of N_{l+1}, and thus between an element $\mathcal{I}(i,j)$ of c_l and an element $\mathcal{O}(i,j)$ of c_{l+1}. In presence of a skip connection of length λ, there is an arc in $A_{l-\lambda}$ for each direct connection between a node of $N_{l-\lambda}$ and a node of N_l, and thus between an element $\mathcal{I}(i,j)$ of $c_{l-\lambda}$ and an element $\mathcal{O}(i,j)$ of c_l of the ResNet.

Similarly, W is the set of weights of G. It consists of a set of subsets $W = \{W_1, W_2, ..., W_{L-1}\}$. A weight $w \in W_l$, $1 \leq l \leq L-1$, is associated with an arc of A_l and thus with a direct connection from an element $\mathcal{I}(i,j)$ to an element $\mathcal{O}(i,j)$ of the corresponding ResNet. w is equal to $g_{mean}(i,j)$ or to $g_{median}(i,j)$, depending on whether the mean or the median is chosen as statistical descriptor.

This way of modeling a ResNet through a set of class networks, each representing a layer of a multilayer network, preserves all the peculiarities of the ResNet. In fact, nodes in a class network represent convolutional operations on a specific image location performed by a ResNet. When filters slide over the input, a node is created for each location. The arcs of a class network represent connections between consecutive and/or non-consecutive layers of the ResNet and allow us to model the input flowing through that network. In fact, an arc connects two nodes in the ResNet only if the output of one of them represents the input of the other, regardless of whether the two nodes are in two consecutive layers or not. In this way, we preserve the locality characteristic of convolutional operations and model the interactions between filters of different convolutional layers. Finally, the weights of the arcs represent the strength of the connection between the corresponding ResNet nodes. This strength is measured simply by computing the mean or median of the convolutional filters applied to an image location.

3.3 Mapping a ResNet into a multilayer network

In this section, we use the class network defined above as the basis of a more complex data structure, namely the multilayer network. The latter allows us to represent, analyze and manipulate a ResNet through the concepts and techniques typical of complex network theory (Kivelä et al., 2014).

Let D be a dataset consisting of t target classes $Cl_1, Cl_2, \ldots, Cl_t$ and let $rnet$ be a ResNet. The multilayer network $\mathcal{G} = \{G^1, G^2, ..., G^t\}$, corresponding to $rnet$, is a set of t class networks, one for each target class of the dataset. In particular, the class network G^k, $1 \leq k \leq t$, corresponds to the k^{th} target class of D. In Figure 9, we show a sample multilayer network $\mathcal{G}$ composed of 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).

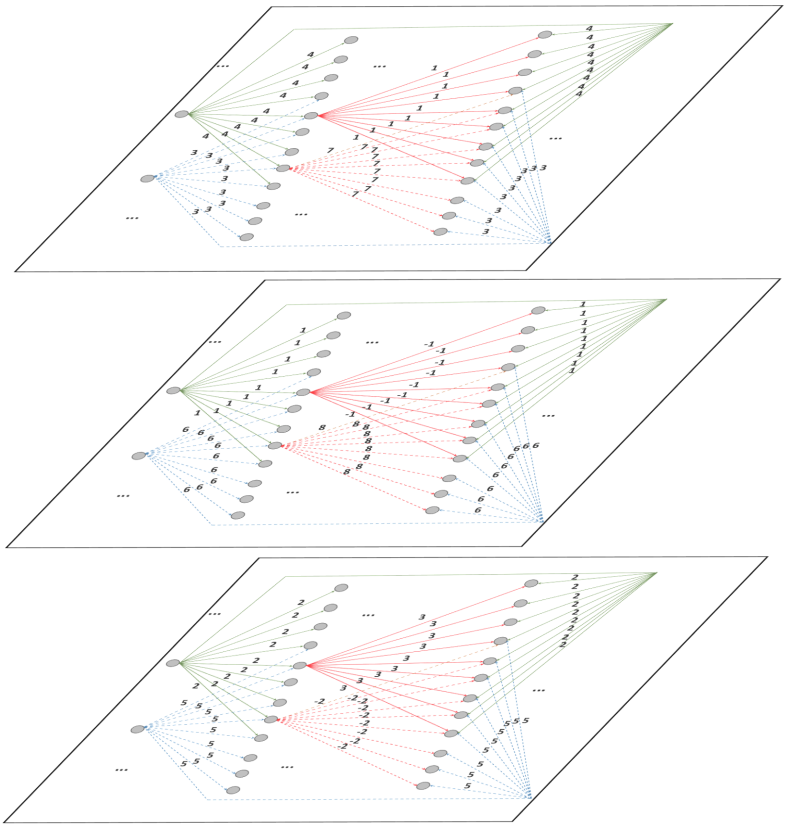
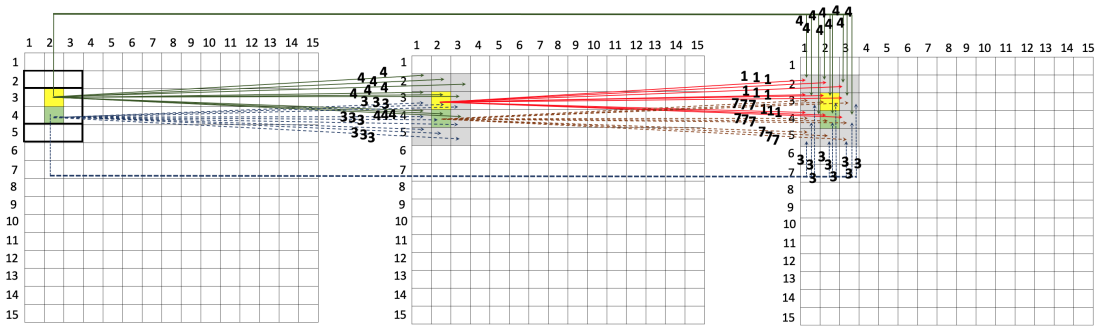


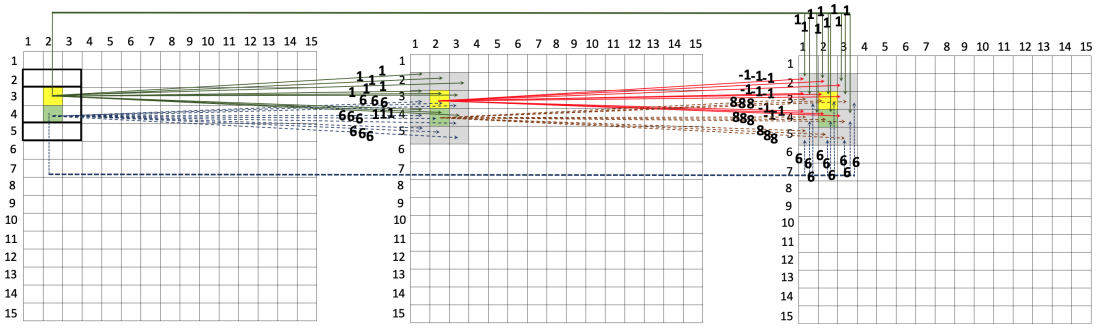
Figure 9: Sample multilayer network $\mathcal{G}$ characterized by three layers corresponding to the class networks G^1 (top), G^2 (middle), and G^3 (bottom)

In Figure 10, we show the creation of a piece of the multilayer network $\mathcal{G}$ depicted in Figure 9. It is generated by extracting the feature maps of three target classes from $rnet$. In it, some direct connections corresponding to some arcs for the class networks of $\mathcal{G}$ are generated by a filter bank of size 3×3 with stride 1, sliding on the positions $(3, 2)$ and $(4, 2)$ of the feature maps. The weights of these arcs are computed as described in Section 3.1, by applying the mean as the statistical descriptor. Specifically, for the class network G^1 (resp., G^2, G^3), two sets of arcs of weights 4 (resp., 1, 2) and 3

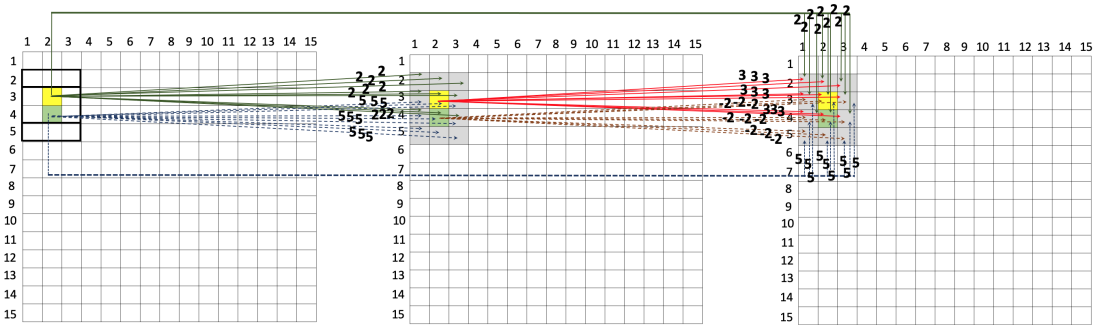
(resp., 6, 5) are created between the first and second feature maps. Furthermore, two sets of arcs of weights 1 (resp., -1, 3) and 7 (resp., 8, -2) are created between the second and the third feature maps. Finally, two sets of skip connections of weights 4 (resp., 1, 2) and 3 (resp., 6, 5) are created between the first and third feature maps.



(a) Class network G^1



(b) Class network G^2



(c) Class network G^3

Figure 10: Creation of a piece of the class networks G^1, G^2 and G^3 representing the layers of the multilayer network $\mathcal{G}$ shown in Figure 9

3.3.1 Algorithm for building the multilayer network

In this section, we illustrate our algorithm for constructing a multilayer network from a ResNet *rnet* and a dataset D consisting of a set of t target classes. The algorithm consists of two steps, namely: *(i)* creation of a list of patch pairs, which represents a support data structure for the next step; *(ii)* creation of the multilayer network from the list of patch pairs. A patch is a part of a feature map; it has the same size as the filters applied by the next convolutional layer and will give rise to a node of the multilayer network. Patch pairs allow us to represent the input and output of a convolutional layer. As it will be clear in the following, the first element of a patch pair is the feature map representing the input of the convolutional layer, while the second element of the pair is the output, obtained by performing the corresponding convolution operation. The idea of using a list of patch pairs makes it easy to handle not only direct connections but also skip connections, which are a specificity of ResNets. Other more rigid data structures, which rely on the coupling of more consecutive convolutional layers, would have been able to handle direct connections very effectively. However, they could not handle skip connections, and thus they could have been used on classical CNNs but not on ResNets.

The function corresponding to the first step of the multilayer network construction, called `CREATE_PATCHES`, is reported in Algorithm 1.

It receives a ResNet *rnet* and a target class Cl_k ($1 \leq k \leq t$) and builds a list of patch pairs. It operates on the feature maps provided as input to each convolutional layer of *rnet*. As previously pointed out, the most important property that makes this function applicable to ResNets is the fact that it operates exclusively on pairs of convolutional layers by creating a patch pair for each layer pair. This feature makes it capable of handling not only direct connections but also skip connections.

`CREATE_PATCHES` proceeds as follows. It uses a list *conv_layers* that initially contains the pairs of consecutive convolutional layers present in *rnet*. It iterates over all the elements of *conv_layers*, providing the images of Cl_k as input. 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 of *target*, which processes it as follows.

At the beginning of each iteration, `CREATE_PATCHES` determines the starting and ending points of the output of *source*. Specifically, 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 performed by *target* to 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. The patches corresponding to a convolutional layer are stored in a list called *patch_list*.

After creating all the patches of a convolutional layer, we store the corresponding *patch_list* in a dictionary of patches called *patch_dict*, whose key consists of the identifiers of the source and target layers. In this way, we can associate a *patch_list* with the convolutional layer that generated it. After that, we proceed to create the possible connections between two *patch_lists*. Recall that, in ResNets, there are not only direct connections but also skip connections. We must take both of them into account when reconstructing the ways in which data flows along the network. To this end, we extract key-value pairs from *patch_dict*, and check whether two *patch_lists* are consecutive (this can happen

Algorithm 1 Function CREATE_PATCHES

Input

- *rnet*: a Residual Neural Network
- Cl_k: a target class
- *get_convolutional_layers*: a function that receives a ResNet and returns the list (*source*, *target*) of pairs of convolutional layers which are consecutive in that model
- *keys*: a function that receives a dictionary and returns its list of keys

Output

- *list_of_patch_pairs*: the list of the consecutive patch pairs

function CREATE_PATCHES()
  patch_dict = {}
  conv_layers = *get_convolutional_layers*(*rnet*)
  for (*source*, *target*) in *conv_layers* **do**
    patch_list = ∅
    $img_{ws} = \left\lfloor \frac{target[\text{“}filter_{width}\text{”}]}{2} \right\rfloor$
    $img_{hs} = \left\lfloor \frac{target[\text{“}filter_{height}\text{”}]}{2} \right\rfloor$
    $img_{we} = source[\text{“}width\text{”}] - img_{ws}$
    $img_{he} = source[\text{“}height\text{”}] - img_{hs}$
    for $i = img_{ws}$ to img_{we} **do**
      for $j = img_{hs}$ to img_{he} **do**
        patch = (*id*, *center*, *width*, *height*, *source*, *target*)
        Add *patch* to *patch_list*
    $patch\_dict[(source[\text{“}id\text{”}], target[\text{“}id\text{”}])] = patch\_list$
  list_of_patch_pairs = ∅
  for (s_1, t_1) in *keys*(*patch_dict*) **do**
    for (s_2, t_2) in *keys*(*patch_dict*) **do**
      if $t_1[\text{“}id\text{”}] == s_2[\text{“}id\text{”}]$ **then**
        Add $(patch\_dict[(s_1[\text{“}id\text{”}], t_1[\text{“}id\text{”}])], patch\_dict[(s_2[\text{“}id\text{”}], t_2[\text{“}id\text{”}])])$ to *list_of_patch_pairs*
  return *list_of_patch_pairs*

because there is a direct connection or a skip connection between the layers from which they are derived). If two *patch_lists* satisfy this condition, we add the corresponding pair to *list_of_patch_pairs*, which represents the output of `CREATE_PATCHES`.

The function corresponding to the second step, called `CREATE_LAYER_NETWORK`, is reported in Algorithm 2. It receives a ResNet *rnet* and a target class Cl_k ($1 \leq k \leq t$), and returns a layer (specifically, the k^{th} layer G^k) 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_pairs* corresponding to *rnet* trained on the images of the class Cl_k. Then, it creates an initially empty network G^k and iterates over *list_of_patch_pairs* considering one pair of *patch_lists* at a time. As specified above, the first list of the pair is considered as *source* while the second list is assumed as *target*.

At the beginning of each iteration, `CREATE_LAYER_NETWORK` adds to G^k a node for each patch present in *source* and a node for each patch present in *target*, if they are not already present therein. Then, it computes the variables w_{ratio} and h_{ratio}. In fact, as we have seen in Section 3.1, a pooling layer

Algorithm 2 Function CREATE_LAYER_NETWORK

Input

- $rnet$: a Residual Neural Network
- Cl_k: a target class
- $get\_feature\_maps$: a function that receives a ResNet and a target class Cl_k and returns the list of the feature maps for each layer of $rnet$ when trained with Cl_k

Output

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

function CREATE_LAYER_NETWORK()
  $f\_maps = get\_feature\_maps(rnet, t)$
  $list\_of\_patch\_pairs = CREATE\_PATCHES(rnet, Cl_k)$
  $G^k = \emptyset$
  for $(source, target)$ in $list\_of\_patch\_pairs$ **do**
    Add nodes from *source* and *target* to G^k 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^k
            $map = f\_maps[source[\text{“}name\text{”}]]$
            Select the portion of *map* 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^k

may exist between two consecutive network layers, which reduces the image size through a pooling operation (e.g., max or average). In that case, the area covered by a filter in *target* is larger than that covered in *source* (see Figure 5).

At this point, `CREATE_LAYER_NETWORK` stores in w_{bound} (resp., h_{bound}) half the width (resp., height) of the filter associated with the convolutional layer. In fact, as we have seen in Section 3.1, the application of a convolutional filter is done with reference to the center of the filter, and therefore with respect to the center 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 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^k.

Once this arc has been inserted, `CREATE_LAYER_NETWORK` computes the corresponding weights by applying the formulas seen in Section 3.1. For this purpose, it first considers the feature map of *source* and selects the portion corresponding 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 $node_t$. Both the weight based on the mean (see Equation 2) and the one based on the median (see Equation 3) are stored for each arc.

After all iterations, `CREATE_LAYER_NETWORK` has created the k^{th} layer G^k, corresponding to the class Cl_k, of the multilayer network $\mathcal{G}$. Calling `CREATE_LAYER_NETWORK` t times, one for each target class of the dataset D, we obtain the final multilayer network.

4 Compressing a ResNet through a multilayer network model

In the previous section, we introduced a method for mapping a ResNet into a multilayer network. It represents the first part of our approach and the first novelty proposed in this paper. The resulting multilayer network can be used to represent, analyze and manipulate the corresponding ResNet. In fact, by applying the concepts and techniques provided by complex network analysis to the multilayer network, it is possible to perform various operations on the ResNet. To give an idea of them, in this section, we describe one of the most relevant ones, which is the compression of a ResNet. It represents the second part of our approach and the second contribution we provide in this paper. This section is structured as follows. First, we present the key aspects of the proposed compression method and supply an example of its behavior. Next, we provide a formal description of it using a pseudocode algorithm.

4.1 Methodology

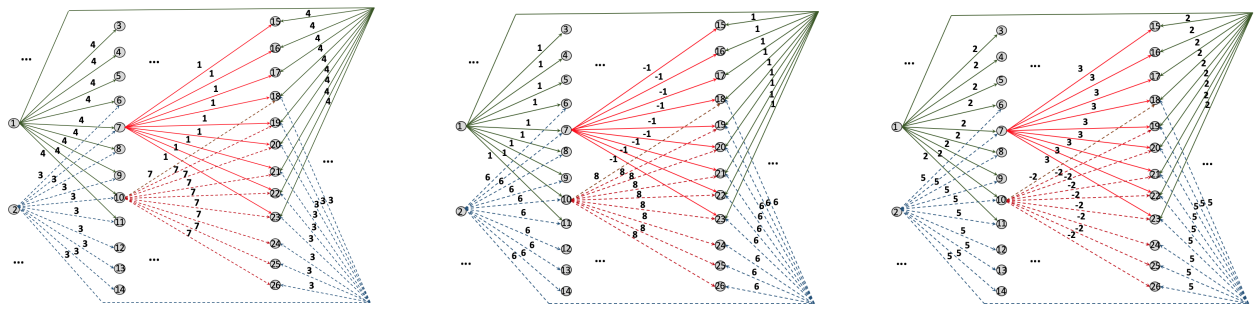
Let $\mathcal{G}$ be a multilayer network and let G^k be its k^{th} layer represented as a class network. For a given node n of G^k, we can define: *(i)* the *indegree* of n, as the sum of the weights of the arcs of G^k incoming into n; *(ii)* the *outdegree* of n, as the sum of the weights of the arcs of G^k outgoing from n; *(iii)* the *degree* of n, as the sum of its indegree and its outdegree. In the following, we use the symbol $d^k(n)$ to indicate the degree of n in G^k, while we use the symbol $\delta(n)$ to denote the overall degree of n in $\mathcal{G}$. As will be clear in the following, $\delta(n)$ is an indicator of the effectiveness of the filters represented by n.

As previously pointed out, a multilayer network $\mathcal{G} = \{G^1, G^2, \ldots, G^t\}$ consists of one layer for each target class. As a consequence, we can obtain the overall degree $\delta(n)$ of the node n in $\mathcal{G}$ by suitably aggregating the degrees $d^1(n), d^2(n), \cdots, d^t(n)$ of n in the t layers of $\mathcal{G}$. In particular, we have:

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

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

In this regard, our approach uses the mean aggregation function for computing the overall degree $\delta(n)$ of n in $\mathcal{G}$:



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

Figure 11: Flattened representation of a portion of the multilayer network $\mathcal{G}$ shown in Figure 9

$$\delta(n) = \frac{\sum_{k=1}^{t} d^k(n)}{t}, \tag{5}$$

The definition of $\delta(n)$ in Equation 5 favors a smoothed distribution of the degree of n in the different layers. This means that a high degree of n in different layers can increase $\delta(n)$. Conversely, the presence of a low degree of n in some layers can penalize $\delta(n)$. The rationale underlying this choice is to favor those nodes whose feature extraction is relevant for different target classes (Gowdra, Sinha, MacDonell, & Yan, 2021). Conversely, it penalizes those nodes that provide a low contribution in some target classes.

Our proposed compression method aims to select a subset of the nodes of $\mathcal{G}$ with the highest values of the overall degree δ. The choice of a suitable subset is used for selecting the best convolutional layers composing the compressed ResNet. Specifically, our method selects the nodes of $\mathcal{G}$ whose values of the overall degree δ are higher than a given threshold:

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

where $\overline{\delta}$ is the mean of the values of the overall degree δ of all the nodes of $\mathcal{G}$. We decided to select the mean statistical operator because it is the one that obtained the best performances. In the formula, γ is a scaling factor in the real interval $[0, +\infty)$ that tunes the contribution of $\overline{\delta}$.

After selecting the subset of the nodes of $\mathcal{G}$ having an overall degree δ greater than th_δ, our method retrieves the set of convolutional layers from which these nodes were extracted and obtains the compressed representation of the ResNet from these layers. Finally, it performs a new training phase to adjust the weights of the compressed network.

In Figure 11, we illustrate a flattened representation of a portion of the multilayer network $\mathcal{G} = \{G^1, G^2, G^3\}$ shown in Figure 9. In that representation, all nodes are numbered from 1 to 26.

In order to determine the threshold th_δ, it is first necessary to compute $\overline{\delta}$. It is equal to the mean of the overall degrees δ of the nodes of $\mathcal{G}$. The value $\delta(n)$ of a node n is computed by applying Equation 5. For instance, $\delta(1)$ is computed as follows:

$$\delta(1) = \frac{[d^1(1) + d^2(1) + d^3(1)]}{3} = \frac{[72 + 18 + 36]}{3} = 42 \tag{7}$$

Here, $d^1(1)$, $d^2(1)$ and $d^3(1)$ are the degrees of node 1 for G^1, G^2 and G^3, respectively.

Analogously, the overall degrees of the other nodes are: $\delta(2) = 84$, $\delta(3) = \delta(4) = \delta(5) = 2.34$, $\delta(6) = \delta(8) = \delta(9) = \delta(11) = 7$, $\delta(7) = 16$, $\delta(10) = 46$, $\delta(12) = \delta(13) = \delta(14) = 4.67$, $\delta(15) = \delta(16) = \delta(17) = 3.34$, $\delta(18) = \delta(19) = \delta(20) = \delta(21) = \delta(22) = \delta(23) = 12.34$, $\delta(24) = \delta(25) = \delta(26) = 9$.

We can observe that $\delta(18) = 12.34 > \delta(24) = 9$, because the degree of node 24 at layers 1 and 3 (i.e., $d^1(24) = 10$ and $d^3(24) = 3$) is lower than the one of node 18 at the same layers (i.e., $d^1(18) = 15$ and $d^3(18) = 8$) – see Figure 11.

The value of $\overline{\delta}$ is computed as follows:

$$\begin{aligned} \overline{\delta} = & \frac{[42 + 84 + (2.34 \cdot 3) + 16 + (7 \cdot 4) + 46 + (4.67 \cdot 3) + (3.34 \cdot 3) + (12.34 \cdot 6) + (9 \cdot 3)]}{26} = \\ & = \frac{348.09}{26} = 13.39 \end{aligned} \tag{8}$$

Now, assume, for instance, that the value of the scaling factor γ is 1.15; then, the threshold th_δ is computed as: $th_\delta = \gamma \cdot \overline{\delta} = 1.15 \cdot 13.39 = 15.40$.

The nodes with degree $\delta > th_\delta$ are 1, 2, 7, 10, which are present in the first and second feature maps (see Figures 10 and 11). Accordingly, the compressed ResNet consists of the first and second convolutional layers of the original ResNet, while the third one is pruned from the model.

Finally, in Table 1 we report the parameters used in the compression approach described above, along with their corresponding meaning.

Parameter	Description
$d^k(n)$	Degree of n in the class network G^k
$\delta(n)$	Overall degree of n in $\mathcal{G}$
$\overline{\delta}$	Mean of the overall degree $\delta(n)$ of all the nodes of $\mathcal{G}$
γ	Scaling factor
th_δ	Threshold for selecting the best performing nodes

Table 1: Overview of the parameters of our compression approach

4.2 Approach formalization

In this subsection, we provide a more formal description of our approach to compress a ResNet, which we described informally in the previous subsection. In Algorithm 3, we report the corresponding pseudocode algorithm.

Our algorithm receives three parameters, namely: *(i)* the ResNet *rnet* to be compressed; *(ii)* the multilayer network $\mathcal{G}$ associated with *rnet* and computed by applying the approach described in Section 3.3; *(iii)* the scaling factor γ.

It also uses some support functions, namely:

- *compute_overall_degrees*, which computes the overall degree δ of each node of $\mathcal{G}$.
- *get_convolutional_layers*, which receives a ResNet and returns the list of its convolutional layers.
- *mean*, which receives the set of overall degrees of the nodes of $\mathcal{G}$ and computes their mean.

Algorithm 3 Function COMPRESS_RESNET

Input

- *rnet*: a ResNet to compress
- $\mathcal{G}$: the multilayer network associated with *rnet*
- γ: a real number representing the scaling factor in the computation of th_δ
- *compute_overall_degrees*: a function that computes the multilayer degree δ for each node of $\mathcal{G}$
- *get_convolutional_layers*: a function that returns the list of the convolutional layers of a ResNet
- *mean*: a function that returns the mean of a list of values
- *next*: a function that takes in input a ResNet and one of its layers l and returns the list of the layers after l
- *type*: a function that takes in input a layer of a ResNet and returns its layer type
- *remove_layers*: a function that prunes a list of layers from a ResNet

Output

- $\overline{rnet}$: a compressed version of *rnet*

```
1: function COMPRESS_RESNET()
2:     saving_layers = ∅
3:     removable_layers = ∅
4:     δ = compute_overall_degrees(G)
5:     th_δ = γ · mean(δ)
6:     for each node n of G do
7:         if δ(n) > th_δ then
8:             Add the convolutional layer associated with n to saving_layers
9:     removable_convolutional_layers = get_convolutional_layers(rnet) \ saving_layers
10:    for each layer l of removable_convolutional_layers do
11:        next_layers = next(rnet, l)
12:        for each layer nl of next_layers do
13:            if (type(nl) = "BatchNormalization") ∨ (type(nl) = "Activation") ∨ (type(nl) = "Add") then
14:                Add nl to removable_layers
15:            else
16:                break
17:    removable_layers = removable_layers ∪ removable_convolutional_layers
18:    rnet' = remove_layers(rnet, removable_layers)
19:    return rnet'
```

- *next*, which receives *rnet* and one of its layers l and returns the layers of *rnet* following l.
- *type*, which receives a layer of *rnet* and returns the corresponding type[1].
- *remove_layers*, which receives *rnet* and a list *removable_layers* of layers to be pruned from it and returns a compressed ResNet $\overline{rnet}$, obtained from *rnet* by pruning the layers of *removable_layers*.

First, `COMPRESS_RESNET` creates two empty lists, namely: *(i) saving_layers*, which contains the layers of *rnet* to be preserved in $\overline{rnet}$, and *(ii) removable_layers*, which contains the layers of *rnet* to be pruned. Then, it calls the function *compute_overall_degrees*, which retrieves the overall degree δ of the nodes of $\mathcal{G}$. Afterwards, it computes the threshold th_δ.

[1]For instance, some possible layer types are "BatchNormalization", "Activation", "Add", "Pooling", and so forth.

At this point, for each node n of $\mathcal{G}$, `COMPRESS_RESNET` checks whether its overall degree $\delta(n)$ is greater than th_δ. In the affirmative case, it adds the convolutional layer associated with n to the list *saving_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 activities of *rnet*. Once all saving layers have been identified, `COMPRESS_RESNET` obtains the prunable layers by calling the function *get_convolutional_layers* with *rnet* as input and removing the layers present in *saving_layers*.

The next step consists in identifying the layers after each convolutional layer present in *removable_convolutional_layers*, because some of them may no longer be useful. In fact, if we remove a convolutional layer, the next activation layer (if any) may be also removed since it can no longer compute the corresponding activation function. A similar reasoning applies to the batch normalization and add layers that are located immediately after a convolutional layer. In fact, also these layers have no reason to exist if the convolutional layer that precedes them is removed. To remove each unnecessary layer, given a convolutional layer that is about to be removed, `COMPRESS_RESNET` checks whether the type of one or more of the layers located after it is "BatchNormalization" or "Activation" or "Add" and puts all the next layers satisfying this condition in the set *removable_layers*. It repeats these tasks for each convolutional layer present in *removable_convolutional_layers*.

After determining the layers to be pruned, `COMPRESS_RESNET` calls the function *remove_layers* and provides it with *rnet* and *removable_layers* as input. This function prunes all *removable_layers* from *rnet* thus obtaining $\overline{rnet}$, which is also the output of `COMPRESS_RESNET`.

4.3 Overview of the compression approach

Figure 12 shows a schematic overview of the main steps of our compression approach. First, the original ResNet is trained on the image dataset. For this purpose, we used the CIFAR-10 dataset. This consists of natural images categorized into 10 target classes. During the forward step, each image in the training set is forwarded through the ResNet to predict the target class. This prediction represents the output of the ResNet. It is compared with the real target class of the image to calculate the error. The latter is used in the backward step to tune back the weights of the convolutional filters and the weights of the fully connected layers in the ResNet.

To create the class network G^i, which represents the i^{th} layer of the multilayer network $\mathcal{G}$, we first forward an image belonging to the i^{th} class Cl_i of the CIFAR-10 dataset through the ResNet. Then, we compute the values of the feature maps derived from the convolutional operations using the pre-computed elements of the convolutional filters. Afterwards, we create nodes from the current feature map and connect them through arcs according to the spatial adjacency in the next feature map. We also model the skip connections of the ResNet as arcs connecting non-consecutive feature maps. The weights of the arcs connecting a node in the current feature map with adjacent nodes in the next feature map correspond to the result of convolving the filter with the area containing that node in the current feature map. This procedure is iterated for a number of times equal to the number t of target classes in the dataset. Since our dataset (i.e., CIFAR-10) has 10 classes, this procedure is iterated 10 times. In this way, 10 class networks $G^1, G^2, \cdots, G^{10}$ are created; they are the graphs representing the layers of the multilayer network $\mathcal{G}$.

In the example reported in Figure 12, we show in detail only the layers G^1 and G^2 of the obtained

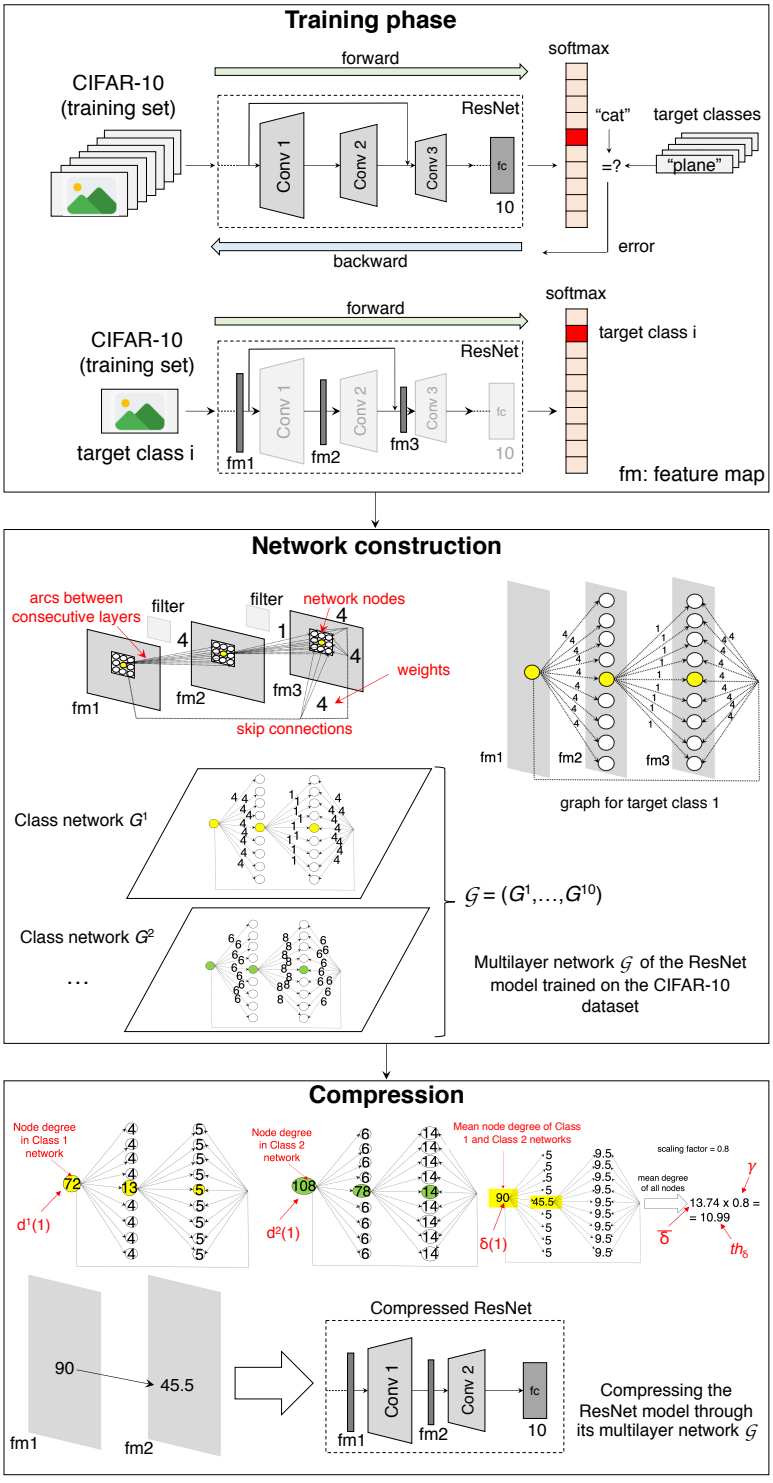


Figure 12: Schematic overview of the main steps of our compression approach

multilayer network $\mathcal{G}$, and illustrate how they behave in the compression of the ResNet modeled by $\mathcal{G}$. Our compression strategy is based on the principle that the nodes of $\mathcal{G}$ having a higher degree

correspond to more informative areas of the feature maps of the ResNet. Accordingly, in the compression procedure, first we need to compute the degree $d^i(j)$ of each node j of $\mathcal{G}$ in each layer G^i of $\mathcal{G}$. For example, in G^1, the degree $d^1(1)$ of the node 1 is 72. It is obtained from the 18 arcs of weight 4 connecting the node 1 with the rest of G^1. Next, for each node j of $\mathcal{G}$, we compute the overall degree $\delta(j)$ as the mean of the degrees of j in the different layers of $\mathcal{G}$. For example, for the node 1 of $\mathcal{G}$, if we consider only the two layers G^1 and G^2, we obtain an overall degree $\delta(1) = 90$, which is the mean of $d^1(1) = 72$ and $d^2(1) = 108$.

Afterwards, we select from $\mathcal{G}$ only those nodes whose overall degree δ is greater than the threshold th_δ. This threshold is obtained by multiplying the mean overall degree $\overline{\delta}$ by a scaling factor γ. In the example shown in Figure 12, $\overline{\delta}$ is obtained as $\overline{\delta} = \frac{90+45.5+(5\times8)+(9.5\times9)}{19} = 13.74$, the scaling factor γ is set to 0.8, so th_δ is equal to 10.99. The nodes having an overall degree greater than th_δ are the ones with overall degree equal to 90 and 45.5. As shown in Figure 12, these nodes are located in the first and second feature maps (i.e., fm_1 and fm_2). Therefore, the feature maps with the highest information content are fm_1 and fm_2 and, as a result, we can compress the ResNet by removing the third convolutional layer.

5 Experiments

5.1 Reference datasets

For our experiments, we adopted two image datasets, namely CIFAR-10 and CIFAR-100[2] (Krizhevsky & Hinton, 2009), which are considered two benchmarks for evaluating deep learning models (K. He et al., 2016a; Han, Kim, & Kim, 2017; B. Li & He, 2018).

CIFAR-10 consists of 60,000 color images of size 32×32 in red, green and blue channels. These images are divided into 10 mutually exclusive and non-overlapping classes, each consisting of 6,000 images. The test set is a single batch consisting of 1,000 images for each class extracted randomly, for a total of 10,000 images. The rest of the images are distributed randomly among five training batches. Some batches may contain a different number of images for each class. Altogether, the training batches contain exactly 5,000 images for each class.

CIFAR-100 is a variant of CIFAR-10 where the number of classes is extended to 100. In this case, each class contains 600 images; 500 of them are used for the training set while 100 images are used for the test set.

5.2 Reference ResNets

In order to build the reference ResNet architecture for CIFAR-10, we followed the ResNet-v2 architecture proposed in (K. He et al., 2016b). Here, there are $2m + 1$ convolutional layers with 16 filters, $3m$ convolutional layers with 64 filters, $3m$ convolutional layers with 128 filters, and m convolutional layers with 256 filters. Each filter is of size 3×3. The top of the network has a global average pooling, followed by a 10-way fully-connected layer with a softmax activation function. In this way, the ResNet consists of a total of $9m + 2$ stacked weighted layers. Following this architecture, we set

[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

m to 2, 6 and 12, to obtain three ResNets, namely ResNet20-v2, ResNet56-v2, and ResNet110-v2, for our experiments. We used the same ResNet architecture for the CIFAR-100 dataset, changing the last dense layer with a 100-way fully-connected layer.

Figure 13 shows the architecture of ResNet20-v2 in terms of convolutional layers, number of filters and skip connections for the CIFAR-10 dataset.

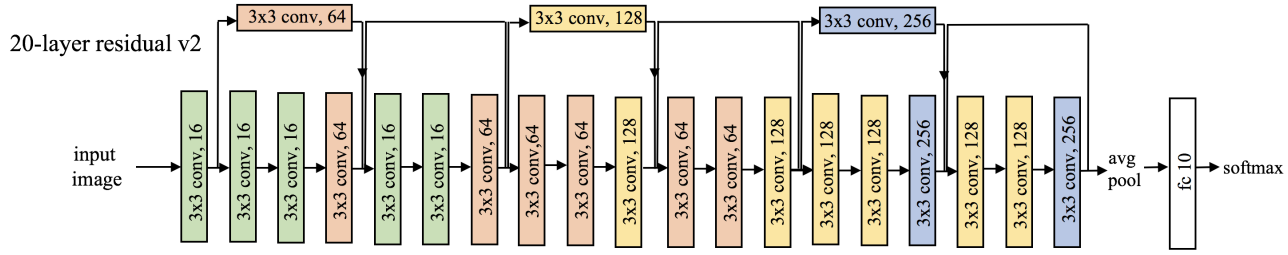


Figure 13: ResNet architecture with 20 weighted layers (ResNet20-v2); *fc* refers to the fully-connected layer

5.3 Obtained results

In this section, we show the results obtained by performing a series of experiments using the free version of Google Colab, which provided a GPU NVIDIA Tesla K80, 12 GB RAM, and two Intel Xeon CPUs 2.30GHz. The programming environment consisted of Python 3.7, Tensorflow 2.6, OpenCV 4.2 and Scikit-learn 1.0.

5.3.1 Tuning and performances

We tested our approach on the ResNets mentioned in Section 5.2 namely ResNet20-v2, ResNet56-v2, and ResNet110-v2. However, we would like to point out that it is general and can be applied to several kinds of ResNet.

In order to evaluate the performance of our approach, we used CIFAR-10 and CIFAR-100. In both datasets, we used 90% of images as training set and 10% of images as validation set. We initially trained the ResNet models from scratch. Then, we built the layers of the multilayer network from the target classes of the training set. For each target class, we computed the means of the feature maps on the image set and adopted these values as the weights of the arcs.

To set the hyperparameters of ResNet20-v2, ResNet56-v2, and ResNet110-v2, for each of these models we randomly selected the hyperparameter values 50 times. Each time we trained the models with the selected hyperparameter values on our dataset. The selection of hyperparameter values was not totally random, but each value was chosen from a specific set. In particular, we selected the batch size in the set [16, 32, 64, 128], the learning rate η in the set [0.0001, 0.001, 0.01, 0.1], the optimizer in the set [Adam, SGD], and the epoch number in the range [100, 1000], with steps of 20. Then, we computed the performance measure of our compression algorithm on the validation set. In the end, we chose the hyperparameter values that guaranteed the best performance measures on the validation set.

They were the same for all the three models. In order to limit overfitting, we checked the validation loss and stopped training when no change in validation loss occurred for five epochs.

In Table 2, we report the hyperparameters, along with their meaning and selected value.

Parameter	Description	Value
Batch size	Number of samples forwarded through the model	128
Learning rate η	Step size for adjusting model weights	0.001
Optimizer	Algorithm for adapting model attributes, like learning rate and weights	Adam
Epoch number	Number of complete iterations through the training data	200

Table 2: Overview of the hyperparameters of the learning procedure

For each pair consisting of a ResNet and a dataset, we reported the six main metrics for measuring the performance of a neural network (i.e., mean epoch time, number of parameters, Top-1 accuracy, F1-score, precision, and recall) against the variation of the value of the parameter γ defined in our compression method (see Equation 6) computed on the test set. The values obtained for two combinations of a ResNet and a dataset against the increase of γ are shown in Figures 14 and 15. The same trends can be observed for the other four combinations. From the analysis of these figures, we can note that, as the value of γ increases, the mean epoch time and the number of parameters have a similar trend. In particular, we observe an initial reduction that is followed by a zone of almost constant values and, subsequently, in presence of very high values of γ, a zone in which these two parameters collapse.

By analyzing these three zones, it is possible to see how our approach provides useful insights regarding the compression of a ResNet. In the first zone, the mean epoch time and the number of parameters decrease considerably while the other metrics fluctuate slightly. This produces variations in the classification metrics that depend on the removed layers. For these values of γ, the ResNet compression is performed without excessively compromising the classification ability of the neural networks. Even, for more complex ResNets, such as those analyzed in Table 3, there is a significant improvement. For intermediate values of γ, there are minimal changes in the metrics. Finally, for high values of γ, there is a collapse of all metrics. This behavior highlights that if too many layers are removed, the network cannot classify properly.

Residual Network	Dataset	γ	Mean epoch time (s)		No. of parameters		Top-1 accuracy		F1-score		Precision		Recall	
			Baseline	Actual value	Baseline	Actual value	Baseline	Actual value	Baseline	Actual value	Baseline	Actual value	Baseline	Actual value
ResNet56-v2	CIFAR-10	1.1	139.16	80.68	1,663,338	1,376,202	0.744	0.816	0.746	0.825	0.745	0.818	0.748	0.832
	CIFAR-100	1.05	139.23	32.86	1,686,468	1,276,644	0.487	0.543	0.487	0.554	0.499	0.558	0.477	0.551
ResNet110-v2	CIFAR-10	0.95	203.30	174.70	3,302,442	2,978,250	0.807	0.813	0.790	0.814	0.780	0.813	0.800	0.815
	CIFAR-100	1.0	206.74	162.21	3,325,572	2,767,524	0.524	0.517	0.521	0.529	0.509	0.527	0.534	0.532

Table 3: Values of the performance metrics obtained for ResNet56-v2 and ResNet110-v2 trained on the CIFAR-10 and CIFAR-100 datasets

In order to better evaluate the benefits of our method, in Figures 14 and 15, for each metric, we also report the values of the corresponding baseline. This represents the values obtained by the same metric calculated on an identical network and dataset but without the application of our compression algorithm.

The first results obtained, shown in Figure 14, concern ResNet20-v2 trained on the CIFAR-10 dataset. Looking at this figure, we can see that, in this case, the best value of γ is 0.15. Considering

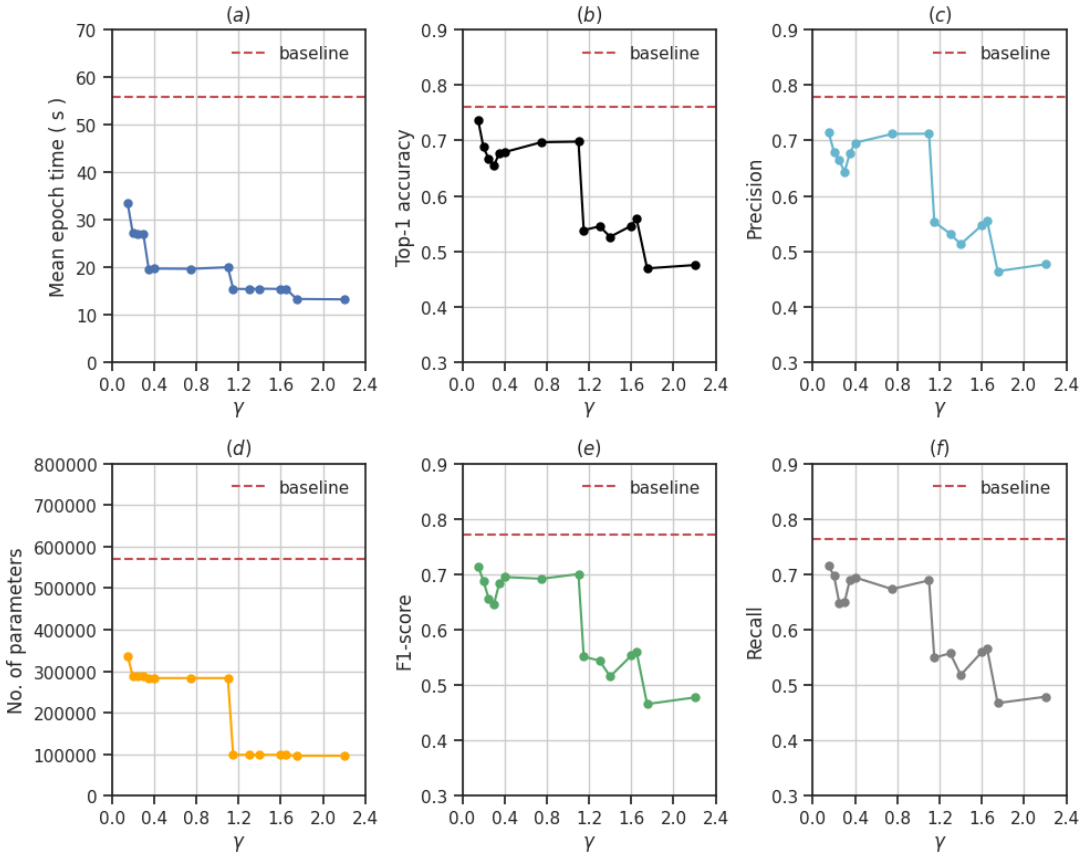


Figure 14: Evaluation of the performance metrics obtained by ResNet20-v2 with the CIFAR-10 dataset

this value, we can observe that the mean epoch time (Figure 14(a)) undergoes a high decrease from 55.75 seconds per epoch of the baseline to 33.50 seconds after applying our approach. Specifically, this decrease is equal to 39.91%. A similar result is obtained by analyzing the number of model parameters (Figure 14(d)), which decreases from $570,602$ to $337,368$, with a decrement of $40,87\%$. In contrast, the value of the Top-1 accuracy (Figure 14(b)) shows only a small decrease of 3.16% with respect to the baseline value; in particular, it is equal to 0.737. The other classification metrics, namely F1-score (Figure 14(e)), precision (Figure 14(c)), and recall (Figure 14(f)) also follow a similar trend. In fact, compared to the baseline values obtained for the original network, they show a decrease of 7.76%, 8.35% and 6.42%, respectively.

The minimal variation of these last four metrics, coupled with the great decrease in mean epoch time and number of parameters, demonstrates the suitability of our approach. Indeed, it significantly reduces the complexity of the network at the expense of only a slight decrease in its classification ability.

Figure 16 shows the convolutional layers of ResNet20-v2 after applying our compression approach. The retained layers are represented with a circle while the pruned layers are denoted with a cross. In this figure, we can see that the growth in the value of γ also leads to a growth in the number of pruned layers, making the ResNet smaller. As discussed previously, the value of γ that provides the

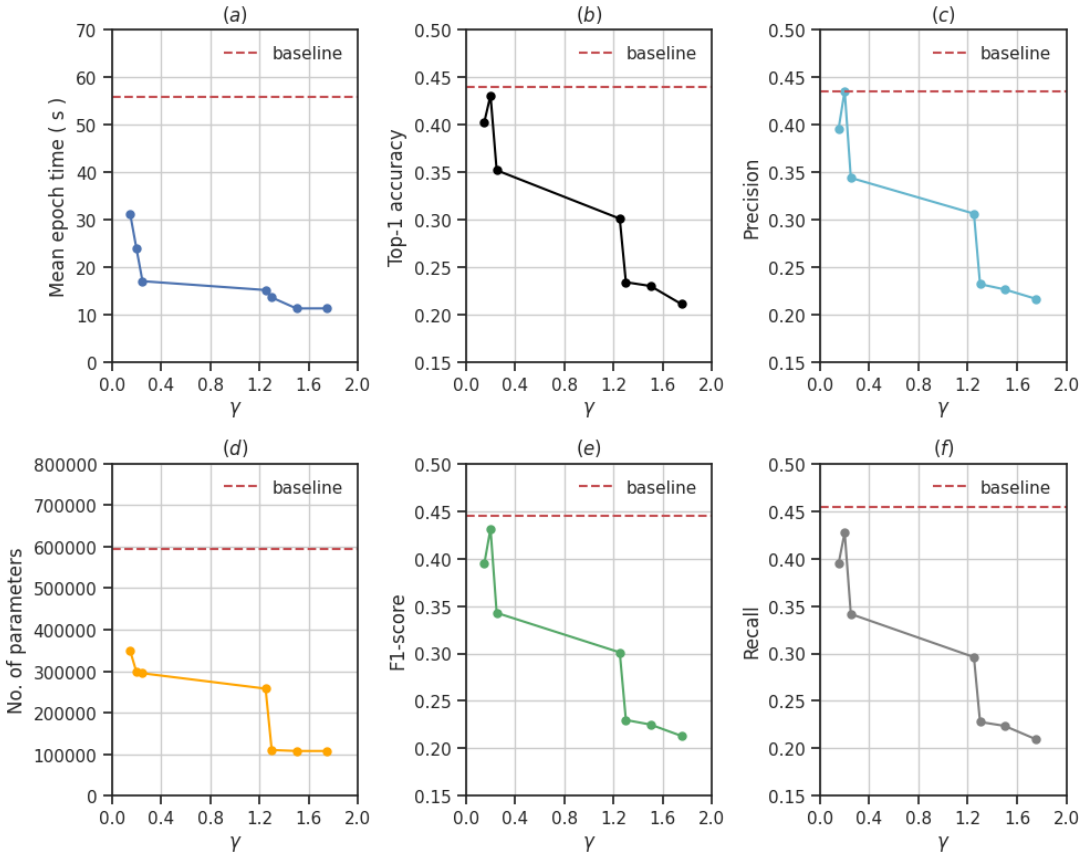


Figure 15: Evaluation of the performance metrics obtained by ResNet20-v2 with the CIFAR-100 dataset

best performance is 0.15. As can be seen from Figure 16, with this value of γ, the network has only two pruned layers located at its end. In particular, they are the last two 3×3 conv, 256 filters.

To further evaluate our algorithm, we again considered ResNet20-v2, but this time we employed the CIFAR-100 dataset. Figure 15 shows the variation of the performance metrics, previously seen for CIFAR-10, against the values of γ. In this case, the best performance is obtained by choosing a value of γ equal to 0.2. With this value, we can see that the mean epoch time (Figure 15(a)) and the number of parameters (Figure 15(d)) have a behavior similar to the previous case, as they decrease by 57.00% and 50.31%, respectively, compared to the baseline values. Figure 15(b) shows that the values of Top-1 accuracy decrease slightly (specifically, by 2.05%) compared to those of the baseline. If we compare the absolute values of this parameter with those obtained under the same conditions in the case of CIFAR-10, we can see that they are lower. However, this is an expected result since CIFAR-100 is more complex than CIFAR-10. The values of F1-score (Figure 15(e)) and recall (Figure 15(f)) are also slightly lower (specifically, by 1.09% and 5.95%, respectively) than those of the baseline. In contrast, the values of precision, shown in Figure 15(c), do not change significantly from those of the baseline.

In Figure 17, we show the convolutional layers preserved and pruned by our approach in this case.

Layer ↓	Dataset →	CIFAR-10														
	γ →	0.15	0.2	0.25	0.3	0.35	0.4	0.75	1.1	1.15	1.3	1.4	1.6	1.65	1.75	2.2
3x3 conv, 16		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
3x3 conv, 16		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
3x3 conv, 16		o	o	o	o	o	o	o	o	o	o	o	o	x	x	x
3x3 conv, 64		o	o	o	o	o	o	x	x	x	x	x	x	x	x	x
3x3 conv, 64		o	o	o	o	x	x	x	x	x	x	x	x	x	x	x
3x3 conv, 16		o	o	o	o	o	o	o	o	o	o	o	o	o	o	x
3x3 conv, 16		o	o	o	o	o	o	o	o	o	o	o	o	o	x	x
3x3 conv, 64		o	o	o	x	x	x	x	x	x	x	x	x	x	x	x
3x3 conv, 64		o	o	o	o	o	o	o	o	o	o	o	x	x	x	x
3x3 conv, 64		o	o	o	o	o	o	o	o	o	x	x	x	x	x	x
3x3 conv, 128		o	o	o	o	o	x	x	x	x	x	x	x	x	x	x
3x3 conv, 128		o	x	x	x	x	x	x	x	x	x	x	x	x	x	x
3x3 conv, 64		o	o	o	o	o	o	o	o	o	o	x	x	x	x	x
3x3 conv, 64		o	o	o	o	o	o	o	o	x	x	x	x	x	x	x
3x3 conv, 128		o	o	x	x	x	x	x	x	x	x	x	x	x	x	x
3x3 conv, 128		o	o	o	o	o	o	o	o	o	o	o	x	x	x	x
3x3 conv, 128		o	o	o	o	o	o	o	x	x	x	x	x	x	x	x
3x3 conv, 256		o	o	o	o	o	x	x	x	x	x	x	x	x	x	x
3x3 conv, 256		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
3x3 conv, 128		o	o	o	o	o	o	o	o	x	x	x	x	x	x	x
3x3 conv, 128		o	o	o	o	o	o	o	o	x	x	x	x	x	x	x
3x3 conv, 256		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 16: Convolutional layers of ResNet20-v2 preserved and pruned by our compression algorithm for CIFAR-10

From the analysis of this figure we can see that our algorithm prunes three convolutional layers from the second half of ResNet20-v2. The removed layers are the ones that contribute the least to the classification performance of the network. Given the analysis of the values of the metrics done in Figure 15, we can see that this compression configuration of the ResNet allows for a strong reduction of its complexity, while keeping its predictive capabilities almost unchanged.

The performance results for the other two ResNets under review are shown in Table 3. In this table, we report the best results for each network. The first examination we did involved ResNet56-v2 trained with the CIFAR-10 dataset. This type of ResNet has a higher complexity than the previous one due to the presence of a higher number of layers. In this case, the best performance results are obtained by setting γ equal to 1.1. This value of γ allows us to obtain a considerable decrease in the average time per epoch (equal to 42.01%) compared to the corresponding baseline value. Instead, the number of model parameters shows a decrease, but smaller, equal to 17.26%. However, we observe that the total number of model parameters in the baseline was 1,663,338. In light of this, the result we obtained can be considered satisfactory, because the absolute number of pruned parameters is very high. As for the other classification metrics, we can see that each of them exceeds its baseline. In other words, in this case, the application of our algorithm not only reduces the execution time and the

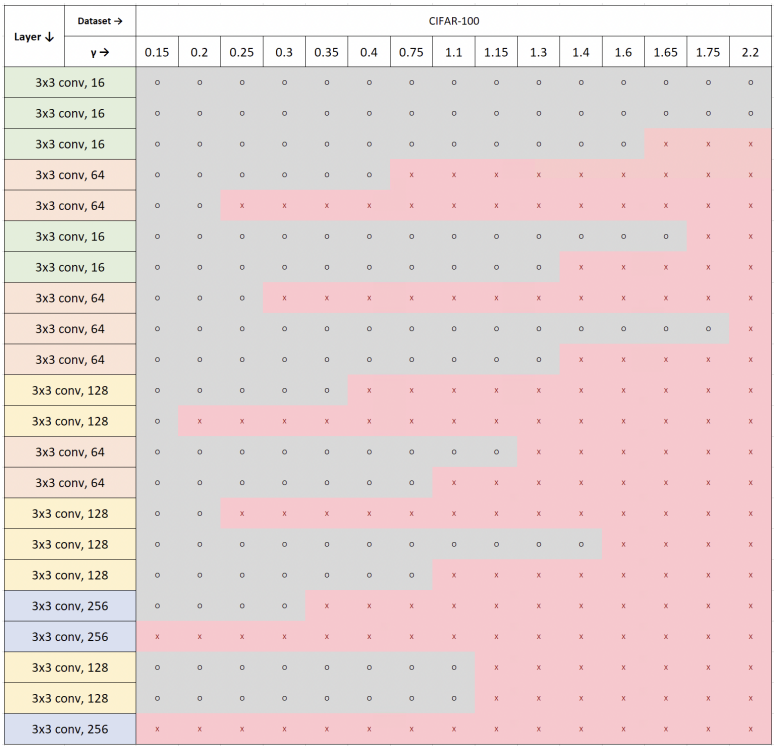


Figure 17: Convolutional layers of ResNet20-v2 preserved and pruned by our compression algorithm for CIFAR-100

number of model parameters of the ResNet on which it is applied, but also increases its classification capabilities. In fact, in this case, we have a Top-1 accuracy of 0.816, compared to the baseline value of 0.744, thus reaching an increase of 9.67%. In addition, F1-score, precision and recall show an increase of 10.59%, 9.80% and 11.23%, respectively, over the corresponding baseline values.

By analyzing the evolution of the metrics of ResNet56-v2 when operating on CIFAR-100, we saw that the optimal value of γ is 1.05 in this case. Therefore, we used this value in the corresponding computations shown in Table 3. As can be seen from this table, the mean epoch time undergoes a drastic decrease, equal to 76.40%, compared with the corresponding baseline value. The number of model parameters also decreases compared to the one of the baseline; in particular, the decrease is equal to 24.30%. Instead, F1-score, Top-1 accuracy, precision and recall have higher values than the corresponding baseline. Specifically, the growths are equal to 13.76%, 11.50%, 11.82% and 15.51%, respectively.

Finally, we analyzed the results obtained by training ResNet110-v2, which represents the largest ResNet we considered. In Table 3, we show the values of the metrics obtained by this network when it is trained on the CIFAR-10 dataset. In this case, we obtained that the optimal value of γ is 0.95. Therefore, the values shown in Table 3 and our next considerations refer to this value. In this case, we had to find the best tradeoff between the reduction of time and memory required by the execution, on the one hand, and the values of classification metrics, on the other hand. Here, we point out that the latter are still always greater than the values of the corresponding baselines. The mean epoch time

shows a reduction of the 14.06% from the corresponding baseline value. Similarly, the reduction in the number of model parameters with respect to the baseline value is 9.82%. The value of Top-1 accuracy remains essentially constant compared to the corresponding baseline value. Finally, the values of F1-score, precision and recall increase by 3.03%, 4,23% and 1.87%, compared to the corresponding baseline values.

By training the previous ResNet with the CIFAR-100 dataset, we found that the optimal value of γ is 1.0. Selecting this value, we obtained the performance results shown in Table 3. From the analysis of this table we can observe that, with respect to the corresponding baseline values, the mean epoch time has a decrease of 21.54%, while the number of model parameters shows a decrease of 16.78%. The classification metrics present values very similar to those of the baseline. In particular, the Top-1 accuracy presents a slight decrease of 1.33%. The values of F1-score and precision show a slight growth of 1.53% and 3.53%, respectively. Finally, recall presents a slight decrease of 0.37%.

As for the layers removed, ResNet56-v2, trained with the CIFAR-10 dataset, resulted in the removal of 5 convolutional layers, most of which belong to the second half of the network. The compression of the same neural network trained with the CIFAR-100 dataset resulted in pruning 8 layers, 75% of which were in the second half of the network. The compression of ResNet110-v2 performed similarly to the previous two ResNets. Specifically, when that network was trained with CIFAR-10, 27 layers were pruned, 77% of which belonged to the final half of the network. Instead, when the same ResNet was trained with CIFAR-100, 12 convolutional layers were pruned, 75% of which were in the second half of the network.

By analyzing the overall results, it is possible to see how our compression algorithm can be effectively applied to different types of ResNets trained on different datasets. We also note that the effectiveness of our approach depends on the value of γ chosen, because it affects the number of convolutional layers to be pruned, thus increasing or decreasing the level of compression. In particular, the value of γ should be selected based on the size of the network to be compressed. In small deep neural networks, e.g., ResNet20-v2, since there are few layers, most of them are needed for classification and will be retained. Therefore, the value of γ that produces the best compression should be sought among the low values, within the range $(0, 0.3]$. On the other hand, when we want to compress a ResNet with a very high number of convolutional layers, such as ResNet56-v2 and ResNet110-v2, we have to choose a higher value of γ because there may be redundant layers that make a marginal contribution and slow down the network by a large amount. In this case, the experimental results suggest that we should look for the best value of γ in the range $[0.9, 1.2]$.

From these analyses, we can see that our compression technique tends to remove layers located in the final part of the network. These layers generally extract very detailed and specific features. In contrast, our technique tends to leave intact the initial layers, placed at the beginning of the network, which extract more general features. In particular, the last convolutional layer of the ResNet always appears among the pruned ones. From this point of view, we can conclude that it is always one of the first layers that can be removed in a compression process.

5.3.2 Comparison with existing pruning approaches

In this section, we propose a comparison between our approach and other related ones available

in the past literature. For this purpose, we consider the pruning methods described in (Chen & Zhao, 2019), (Shao, Dai, Kuang, & Meng, 2021), (Ayinde & Zurada, 2018), (C. Zhang et al., 2021), as well as the two pruning methods present in (H. Li et al., 2016), which are referred to as Methods "A" and "B" in that paper. In these papers, the authors report the accuracy and number of model parameters of the ResNet56-v2 and ResNet110-v2 before and after the pruning operation on the CIFAR-10 and CIFAR-100 datasets. We do not consider compression approaches based on knowledge distillation and quantization (Ghimire et al., 2022; Polino, Pascanu, & Alistarh, 2018) instead of pruning because the evaluation of the changes in classification accuracy and parameter decrease caused by them is not straightforward. As a first task of this comparison, in Table 4 we report the qualitative characteristics of our approach and the related ones based on pruning mentioned above. The qualitative characteristics considered are pruning type, re-training type, pruning rate, and pruning after training, as suggested in (Choudhary et al., 2020).

Approach	Pruning type	Re-training type	Pruning rate	Pruning after training
Ours	Layers	Fine-tune	Fixed	Yes
(Chen & Zhao, 2019)	Layers	Knowledge distillation	Fixed	Yes
(H. Li et al., 2016) (Method A)	Filters	Fine-tune	Fixed	Yes
(H. Li et al., 2016) (Method B)	Filters	Fine-tune	Fixed	Yes (iteratively)
(Shao et al., 2021)	Filters	Fine-tune	Dynamic	Yes
(Ayinde & Zurada, 2018)	Filters	Fine-tune	Fixed	Yes
(C. Zhang et al., 2021)	Filters	Not necessary	Fixed	No

Table 4: Qualitative comparison between our ResNet compression approach and several related ones based on pruning

From the analysis of this table, we can observe several differences between the approaches into comparison. In fact, as for the pruning type, (Chen & Zhao, 2019) and our approach prune convolutional layers, while the approaches of (Shao et al., 2021), (Ayinde & Zurada, 2018), (H. Li et al., 2016) and (C. Zhang et al., 2021) prune single filters present in a CNN. Regarding re-training, five of the seven approaches considered perform a fine-tuning of the weights after the pruning operation. Instead, (Chen & Zhao, 2019) leverages knowledge distillation, and (C. Zhang et al., 2021) does not need a re-training process. As for the pruning rate, (Shao et al., 2021) is the only approach providing a method to set this parameter dynamically by measuring the dispersion of each convolutional layer. All the other approaches use a fixed threshold, which is precomputed and then employed to prune the CNN. We point out that we have considered both the methods of (H. Li et al., 2016) as having a fixed pruning rate since the authors define the number of filters to be pruned beforehand. Finally, regarding the last qualitative characteristic considered in Table 4, six of the seven approaches prune the CNN after training it, while (C. Zhang et al., 2021) is able to prune the filters during the training of the model. The (Method B) in (H. Li et al., 2016) prunes filters layer by layer and iteratively retrains the model afterward. In this way, the model is retrained before pruning the next layer to adapt to the changes in the pruning process.

In order to make a fair comparison between the considered approaches, we computed the variation in the classification accuracy and the number of parameters of the input model before and after performing the pruning operation. Specifically, we computed the percentage variation of Top-1 accuracy (which we will denote by Top-1 Accuracy %) as the ratio, expressed as a percentage, of the variation

of this parameter caused by compression to the Top-1 accuracy of the baseline. Similarly, we calculated the percentage variation of the number of pruned parameters (which we will denote by Pruned Parameters %) as the ratio, expressed as a percentage, of the reduction in the number of parameters caused by compression and the number of parameters of the baseline. The comparisons of pruning methods for the CIFAR-10 and CIFAR-100 datasets are provided in Tables 5 and 6, respectively.

Architecture	Reference model	Top-1 Accuracy %	Pruned Parameters %
ResNet56-v2	(Chen & Zhao, 2019)	0.06	42.30
	(H. Li et al., 2016) (Method A)	0.06	10.40
	(H. Li et al., 2016) (Method B)	0.02	27.60
	(Shao et al., 2021)	-0.13	52.70
	(Ayinde & Zurada, 2018)	-0.27	27.00
	(C. Zhang et al., 2021)	-0.36	59.17
	Our approach ($\gamma = 1.10$)	9.67	17.26
	Our approach ($\gamma = 1.30$)	0.80	37.29
	Our approach ($\gamma = 1.35$)	-2.28	63.95
ResNet110-v2	(H. Li et al., 2016) (Method A)	0.02	15.90
	(H. Li et al., 2016) (Method B)	-0.25	38.60
	(Ayinde & Zurada, 2018)	-0.38	39.00
	Our approach ($\gamma = 0.95$)	0.74	9.82
	Our approach ($\gamma = 1.00$)	-9.79	19.90
	Our approach ($\gamma = 1.05$)	-5.08	55.73

Table 5: Comparison of pruning approaches in terms of Top-1 Accuracy % and Pruned Parameters % with the ResNet56-v2 and ResNet110-v2 models operating on the CIFAR-10 dataset

Architecture	Reference model	Top-1 Accuracy %	Pruned Parameters %
ResNet56-v2	(Chen & Zhao, 2019)	-0.34	36.10
	Our approach ($\gamma = 0.25$)	1.03	17.71
	Our approach ($\gamma = 1.05$)	11.50	24.30
	Our approach ($\gamma = 1.10$)	8.62	37.46

Table 6: Comparison of pruning approaches in terms of Top-1 Accuracy % and Pruned Parameters % with the ResNet56-v2 model operating on the CIFAR-100 dataset

From the analysis of Table 5, which reports the results for the CIFAR-10 dataset, we can observe that our approach achieves similar or better performance than other pruning approaches proposed in the past literature. For instance, as for ResNet56-v2, when we set γ to 1.1 in our approach, we can observe an increase of 9.67% in the values of Top-1 Accuracy and a decrease of 17.26% in the number of model parameters. We also observe that, with a reduction of around 30% - 40% of the number of model parameters, our approach obtains an increase of the Top-1 accuracy equal to 0.80%, which is higher than the 0.06% obtained by (Chen & Zhao, 2019) and than the 0.02% obtained by (H. Li et al., 2016). On the other hand, (Shao et al., 2021; Ayinde & Zurada, 2018; C. Zhang et al., 2021) prune a high percentage of parameters, which then leads to a degradation of the Top-1 Accuracy.

As for ResNet110-v2, we observe that the results of our approach are in line with those obtained by (H. Li et al., 2016) and (Ayinde & Zurada, 2018). In fact, our approach achieves a decrease of 9.82% in the number of model parameters while obtaining an increase of 0.74% in the value of Top-1 accuracy. In comparison, the approach of (H. Li et al., 2016) achieves an increase of 0.02% in the value of Top-1 accuracy with a decrease of 15.50% in the number of model parameters. On the other

hand, if we want to decrease the number of pruned parameters by about 20% - 50% in our approach, the corresponding Top-1 Accuracy decreases too, and this decrease ranges from 5% to 10%.

From the analysis of Table 6, which shows the results obtained by our approach and the one of (Chen & Zhao, 2019) when operating on the CIFAR-100 dataset, we can see that our approach achieves good results compared to the ones of the approach of (Chen & Zhao, 2019). In fact, in all the cases reported, our approach is able to increase the Top-1 accuracy by a value ranging from 1.03% to 11.5%. It is also capable of reducing the number of model parameters by a value ranging from 17.71% to 37.46%. Furthermore, considering a similar value for the decrease of the number of model parameters (i.e., around 36% - 37%), our approach obtains an increase of 8.62% in Top-1 accuracy, while the approach of (Chen & Zhao, 2019) obtains a decrease of 0.34% in the same parameter.

5.4 Discussion

In this paper, we have proposed a method that maps a ResNet into a multilayer network. Thanks to this mapping, we can apply the theory of complex networks on the multilayer network to analyze and manipulate the corresponding ResNet. In addition, we have presented a compression algorithm that employs the support multilayer network to identify which layers of a ResNet can be removed without causing particularly negative influences on its classification accuracy. We have shown that multilayer network is a model sufficiently rich to map ResNets characterized by the presence of skip connections that link non-consecutive convolutional layers. It is worth noting that the principle behind Algorithms 1 and 2 of this paper, namely those defining the mapping of a ResNet into a multilayer network, can be applied to other types of CNNs, such as LeNet, AlexNet, GoogLeNet and VGG19. In fact, in all these cases, convolutional layers can be mapped through lists of patches allowing the creation of input-output pairs according to the CNN architecture. In this way, we are not limited to dealing only with sequential models, but can handle any connection between layers and, consequently, any data flow in a CNN. Our compression algorithm prunes some of the layers of the input ResNet, which implies that the compressed ResNet has fewer parameters, and thus requires shorter training and inference times. With regard with the choice of compressing a ResNet by pruning some of its layers, we observe that it was shown that pruning a whole layer does not disrupt the nature of a ResNet, as it might happen if single connections were cut.

After mapping a ResNet into a multilayer network, we can think of using the concepts and approaches of network analysis to understand what is happening under the hood of the corresponding ResNet. Continuing in this vein, we might consider developing other approaches or measures that work on a multilayer network mapping a CNN. Think, for instance, of a feature importance measure, which aims to identify the most important parts of a feature map, or a measure to study the evolution of weights for different classes.

Although our approach has several interesting properties, it also presents some limitations. For example, it maps the filters of a convolutional layer as a single unit by applying the mean and median operators on its feature maps. This way of proceeding keeps the size of the multilayer network low, but it may lose interesting information about the single filters. This implies that our approach cannot remove a single filter from a convolutional layer because it does not represent the corresponding data within the multilayer network model. This limitation is due to the need to find a tradeoff between

the computational time required to create and handle a multilayer network and the benefits that can be gained from a deep analysis. Clearly, if we create a larger multilayer network, we would be able to later develop an appropriate compression algorithm (similar to the one proposed in this paper) to identify the best performing filters associated with the convolutional layers of the ResNet.

6 Conclusion

In this paper, we have presented a method to map a ResNet into a multilayer network. This representation keeps all the peculiarities of the ResNet model, such as convolutional layers, represented through the nodes of the multilayer network, and direct and skip connections between convolutional layers, represented through the arcs of the multilayer network. After the mapping method, we presented an appropriate pruning algorithm capable of removing some layers of the source ResNet to obtain a compressed version of it. We tested our mapping method and compression algorithm on three types of ResNets (i.e., ResNet20-v2, ResNet56-v2 and ResNet110-v2) and two benchmark datasets (i.e., CIFAR-10 and CIFAR-100). Our experiments showed that our approach is capable of producing a pruned ResNet with fewer model parameters but still able to guarantee good values of accuracy, precision and recall.

This paper is certainly a point of arrival for some of our research efforts, but it is also a starting point for future activities. In fact, we plan to extend our approach so that we can develop a measure of feature importance, which could be useful for quantitatively defining the most important feature map areas for the classification of an input image. In addition, we would like to move beyond using convolutional filters as a single unit. Last but not least, it would be interesting to apply this approach to Natural Language Processing tasks (i.e., text classification, keyword extraction), in which different types of CNNs are employed, and then investigate its effectiveness.

References

Abbasi-Asl, R., & Yu, B. (2017). Structural compression of convolutional neural networks. *arXiv preprint arXiv:1705.07356*.

Aghli, N., & Ribeiro, E. (2021). Combining Weight Pruning and Knowledge Distillation for CNN Compression. In *Proc. of the international conference on computer vision and pattern recognition (cvpr'21)* (pp. 3191–3198). virtual event.

Amini, N., & Zhu, Q. (2022). Fault detection and diagnosis with a novel source-aware autoencoder and deep residual neural network. *Neurocomputing*, *488*, 618–633. (Elsevier)

Ayinde, B., & Zurada, J. (2018). Building efficient convnets using redundant feature pruning. *arXiv preprint arXiv:1802.07653*.

Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade* (pp. 421–436). (Springer)

Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, *3*(1), 1–122. (Now Publishers, Inc.)

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

Choudhary, T., Mishra, V., Goswami, A., & Sarangapani, J. (2020). A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, *53*(7), 5113–5155.

Damacharla, P., Rao, A., Ringenberg, J., & Javaid, A. Y. (2021). TLU-Net: A Deep Learning Approach for Automatic Steel Surface Defect Detection. In *Proc. of the international conference on applied artificial intelligence (icapai'21)* (p. 1-6). virtual event.

Dobbs, T., & Ras, Z. (2022). On art authentication and the Rijksmuseum challenge: A residual neural network approach. *Expert Systems with Applications*, *200*, 116933. (Elsevier)

Farooq, M., & Hafeez, A. (2020). Covid-resnet: A deep learning framework for screening of COVID19 from radiographs. *arXiv preprint arXiv:2003.14395*.

Gao, K., Huang, L., & Zheng, Y. (2021). Fault detection on seismic structural images using a nested residual U-Net. *IEEE Transactions on Geoscience and Remote Sensing*, *60*, 1–15. (IEEE)

Ghimire, D., Kil, D., & Kim, S. H. (2022). A Survey on Efficient Convolutional Neural Networks and Hardware Acceleration. *Electronics*, *11*(6), 945. (MDPI)

Gowdra, N., Sinha, R., MacDonell, S., & Yan, W. Q. (2021). Mitigating severe over-parameterization in deep convolutional neural networks through forced feature abstraction and compression with an entropy-based heuristic. *Pattern Recognition*, *119*, 108057. (Elsevier)

Han, D., Kim, J., & Kim, J. (2017). Deep pyramidal residual networks. In *Proc. of the international conference on computer vision and pattern recognition (cvpr'17)* (pp. 5927–5935). Honolulu, HI, USA.

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep Residual Learning for Image Recognition. In *Proc. of the international conference on computer vision and pattern recognition (cvpr'16)* (p. 770-778). Las Vegas, NV, USA. doi: 10.1109/CVPR.2016.90

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Identity mappings in deep residual networks. In *Proc. of the european conference on computer vision (eccv'16)* (pp. 630–645). Amsterdam, The Netherlands.

He, Y., Lin, J., Liun, Z., Wang, H., Li, L. J., & Han, S. (2018). Amc: Automl for model compression and acceleration on mobile devices. In *Proc. of the european conference on computer vision (eccv'18)* (pp. 784–800). Munich, Germany.

He, Y., Zhang, X., & Sun, J. (2017). Channel pruning for accelerating very deep neural networks. In *Proc. of the international conference on computer vision (iccv'17)* (pp. 1389–1397). Venice, Italy.

Huang, G., Liu, Z., Maaten, L. V. D., & Weinberger, K. (2017). Densely Connected Convolutional Networks. In *Proc. of the international conference on computer vision and pattern recognition (cvpr'17)* (p. 2261-2269). Honolulu, HI, USA: IEEE Computer Society.

Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *Proc. of the european conference on computer vision (eccv'16)* (pp. 646–661). Amsterdam, The Netherlands.

Kang, J., Zhang, J., Li, W., & Zhuo, L. (2021). Crowd activity recognition in live video streaming via 3D-ResNet and region graph convolution network. *IET Image Processing*, *15*(14), 3476–3486.

Kim, H., Jung, W., Park, Y., Lee, J., & Ahn, S. (2022). Broken stitch detection method for sewing operation using CNN feature map and image-processing techniques. *Expert Systems with Applications*, *188*, 116014. (Elsevier)

Kim, H., Khan, M., & Kyung, C. (2019). Efficient neural network compression. In *Proc. of the international conference on computer vision and pattern rrecognition (cvpr'19)* (pp. 12569–12577). Long Beach, CA, USA.

Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J. P., Moreno, Y., & Porter, M. A. (2014). Multilayer networks. *Journal of complex networks*, *2*(3), 203–271.

Koratana, A., Kang, D., Bailis, P., & Zaharia, M. (2019). Lit: Learned intermediate representation training for model compression. In *Proc. of the international conference on machine learning (icml'19)* (pp. 3509–3518). Long Beach, CA, USA.

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images* (Tech. Rep.). Toronto, Ontario, Canada: University of Toronto.

Kumar, K., Prasad, A., & Metan, J. (2022). A hybrid deep CNN-Cov-19-Res-Net Transfer learning architype for an enhanced Brain tumor Detection and Classification scheme in medical image processing. *Biomedical Signal Processing and Control*, *76*, 103631. (Elsevier)

Li, B., & He, Y. (2018). An improved ResNet based on the adjustable shortcut connections. *IEEE Access*, *6*, 18967–18974. (IEEE)

Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2016). Pruning filters for efficient convnets. In *Proc. of the international conference on learning representations (iclr'17).* Toulon, France.

Liu, X., Ma, R., Zhao, J., Song, J., Zhang, J., & Wang, S. (2021). A clinical decision support system for predicting cirrhosis stages via high frequency ultrasound images. *Expert Systems with Applications*, *175*, 114680. (Elsevier)

Luo, J., & Wu, J. (2017). An entropy-based pruning method for cnn compression. *arXiv preprint arXiv:1706.05791*.

Luo, J., Wu, J., & Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. In *Proc. of the international conference on computer vision (iccv'17)* (pp. 5058–5066). Honolulu, HI, USA.

Ma, X., Yuan, G., Lin, S., Li, Z., Sun, H., & Wang, Y. (2019). Resnet can be pruned 60×: Introducing network purification and unused path removal (p-rm) after weight pruning. In *Proc. of the international symposium on nanoscale architectures (nanoarch'19)* (pp. 1–2). Qingdao, China.

Minnehan, B., & Savakis, A. (2019). Cascaded projection: End-to-end network compression and acceleration. In *Proc. of the international conference on computer vision and pattern recognition (cvpr'19)* (pp. 10715–10724). Long Beach, CA, USA.

Polino, A., Pascanu, R., & Alistarh, D. (2018). Model compression via distillation and quantization. In *Proc. of the international conference on learning representations, (iclr'18).* Vancouver, British Columbia, Canada.

Ronald, M., Poulose, A., & Han, D. S. (2021). iSPLInception: an inception-ResNet deep learning architecture for human activity recognition. *IEEE Access*, *9*, 68985–69001. (IEEE)

Shao, M., Dai, J., Kuang, J., & Meng, D. (2021). A dynamic CNN pruning method based on matrix similarity. *Signal, Image and Video Processing*, *15*(2), 381–389. (Springer)

Veit, A., Wilber, M., & Belongie, S. (2016). Residual networks behave like ensembles of relatively shallow networks. In *Proc. of the international conference on neural information processing systems (nips'16)* (p. 550–558). Barcelona, Spain.

Wang, W., Sun, Y., Eriksson, B., Wang, W., & Aggarwal, V. (2018). Wide compression: Tensor ring nets. In *Proc. of the international conference on computer vision and pattern recognition (cvpr'18)* (pp. 9329–9338). Salt Lake City, UT, USA.

Xie, S., Girshick, R., Dollár, P., Tu, X., & He, K. (2017). Aggregated residual transformations for deep neural networks. In *Proc. of the international conference on computer vision and pattern recognition (cvpr'17)* (pp. 1492–1500). Honolulu, HI, USA.

Yoo, H., Han, S., & Chung, K. (2021). Diagnosis Support Model of Cardiomegaly Based on CNN Using ResNet and Explainable Feature Map. *IEEE Access*, *9*, 55802-55813. (IEEE)

Zeng, F., Li, Y., Jiang, Y., & Song, G. (2021). A deep attention residual neural network-based remaining useful life prediction of machinery. *Measurement*, *181*, 109642. (Elsevier)

Zhang, C., Xu, K., Liu, J., Zhou, Z., Kang, L., & Ye, D. (2021). Identity-linked Group Channel Pruning for Deep Neural Networks. In *Proc. of international joint conference on neural networks (ijcnn'21)* (pp. 1–9). (IEEE)

Zhang, H., Jiang, L., & Li, C. (2021). Cs-resnet: cost-sensitive residual convolutional neural network for PCB cosmetic defect detection. *Expert Systems with Applications*, *185*, 115673. (Elsevier)

Zhou, X., Li, X., Hu, K., Zhang, Y., Chen, Z., & Gao, X. (2021). ERV-Net: An efficient 3D residual neural network for brain tumor segmentation. *Expert Systems with Applications*, *170*, 114566. (Elsevier)