POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Politecnico
di Torino

Master's Degree Thesis

On the analysis of the criticalities of deep neural network

Supervisors

Prof. Edgar Ernesto SANCHEZ SANCHEZ

Ph.D. Annachiara RUOSPO

Candidate

Ilaria BRAGAGLIA

Abstract

In recent years, the complexity of emerging computing systems has led to the need of developing new algorithms and solutions for both industry and academy. In this scenario, due to their outstanding computational capabilities, artificial neural networks (ANNs) have become attractive solutions in tasks such as image classification, such as radars, flight control, robots, self-driving cars, space applications. However, to adopt this solution safely in human context, it is crucial to assess their reliability. This work presents a methodology to identify the most critical neurons of a convolutional neural networks (CNN) (Resnet-18) trained on two different datasets, i.e. MNIST and CIFAR-10. This method is based on two levels of analysis: first, the neuron is viewed as an element of each output class (class-oriented analysis); second, the same is interpreted as belonging to the entire neural network (network-oriented analysis). In detail, a neuron that contributes more to the final prediction of the network and therefore carries more information than others, was considered critical. The proposed methodology has been validated by means of two software fault injection campaigns, through which we confirmed that the accuracy of the network decreased much faster in the case of fault involving neurons defined as critical with respect to neurons of minor importance within the network. Subsequently, to improve the reliability of the target convolutional neural networks, a strategy based on the Triple Modular Redundancy (TMR) technique was adopted. The goal was to identify a mitigation technique able to protect all the neurons labeled as critical in the previous step in case they were subject to a fault. The TMR is a fault masking technique that, in our study, allowed to triple all those critical neurons and, through the usage of a majority voter, propagate in the successive layers of the network the most common output of the said neuron, masking possible single faults. In particular, through software fault injection campaigns on different percentages of neurons , we have applied the TMR technique on all the neurons deemed critical and we have shut off all the outputs of those neurons considered not important for the elaboration of the final forecast by the network. Using this method of shutdown parallel to the tripling, we have investigated the trade-off between memory footprint and reliability.

ACKNOWLEDGMENTS

A mio Nonno Mario,
saggio come un gufo,
amorevole come un angelo,
mia dolce bussola per quando mi sono persa.

Table of Contents

List of Figures I

List of Tables II

Acronyms III

1 Introduction 1

2 Background 4
2.1 Neural Networks . . . 4
2.1.1 Models of Biological and Artificial Neurons . . . 4
2.1.2 Artificial Neural Networks Architectures . . . 6
2.2 Activation Functions . . . 6
2.2.1 Linear activation function . . . 7
2.2.2 Non-Linear Activation Functions . . . 7
2.3 ANN learning process . . . 10
2.4 ResNet-18 . . . 11
2.4.1 Convolutional Neural Networks . . . 11
2.4.2 Residual Neural Network, intuition and improvements . . . 13
2.4.3 ResNet-18 architecture . . . 14
2.4.4 Software-level Fault Models in Deep Neural Networks . . . 15

3 Proposed Approach 17
3.1 Identification of the Criticality of Single Neurons . . . 17
3.2 Triple Modular Redundancy on Artificial Neural Network. . . . 19

4 Case study 22

5 Experimental setup analysis 26
5.1 Critical Neurons Identification Experimental Setup . . . 26
5.1.1 Class-Oriented Analysis (CoA) Setup . . . 26
5.1.2 Network-oriented Analysis (NoA) Setup . . . 27

5.1.3 Final Network-Oriented list Setup 27
5.2 Triple Modular Redundancy Experimental Setup 29

6 Experimental result analysis **31**
6.1 Critical Neuron Identification Experimental Results 31
6.1.1 Class-Oriented Analysis (CoA) Experimental results 31
6.1.2 Final Network-Oriented list Experimental Results 35
6.2 Triple Modular Redundancy Experimental Results 37
6.3 TMR Memory Footprint Drawback 39

7 Conclusions **40**

Bibliography

List of Figures

1.1 Common ANN architecture. 2

2.1 A biological neuron in comparison to an artificial neural network: (a) human neuron; (b) artificial neuron; (c) biological synapse; and (d) ANN synapses. 5
2.2 Feed-forward neural network 6
2.3 Linear Activation function. 7
2.4 Sigmoid Activation function. 8
2.5 Tanh Activation function. 9
2.6 ReLu Activation function. 10
2.7 2d Convolution operation. 11
2.8 Pooling layer, a practical example. 12
2.9 Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks.[15] 13
2.10 ResNet Skip connection. [15] 13
2.11 ResNet-18 Architecture 14
2.12 ResNet-18 Basic Block 15

3.1 The critical neuron identification process : a pratical example with CIFAR-10 dataset 19
3.2 TMR implementation on a critical neuron 20

5.1 A Practical example of identifying the neurons to be tripled 29

6.1 Class-Oriented analysis on MNIST 33
6.2 Class-Oriented analysis on CIFAR-10 34
6.3 Final Network-Oriented Analysis on MNIST 36
6.4 Final Network-Oriented Analysis on CIFAR-10 36
6.5 37
6.6 38

List of Tables

4.1 MNIST and CIFAR-10 Image Distribution 22
4.2 ResNet-18 architecture trained on MNIST 23
4.3 ResNet-18 architecture trained on CIFAR-10 24
4.4 ResNet-18 benchmarks. 25

Acronyms

AI artificial intelligence

NN Neural Network

ANN Artificial Neural Network

DNN Deep Neural Network

Tanh Tangent Hyperbolic

ReLu Rectified Linear Unit

CNN Convolutional Neural Networks

FC Fully Connected

FI Fault injection

DP Deep Learning

ML Machine Learning

ResNet Residual Neural Network

BatchNorm Batch Normalization

CoA Class-oriented analysis

NoA Network-oriented analysis

TMR Triple Modular Redundancy

Chapter 1

Introduction

Nowadays, artificial neural network (ANN)-based solutions are spreading more and more in safety-critical applications across different domains, due to their outstanding capabilities. Those brain-inspired computing system were developed to be able to mimic complex features of the biological human model. The human nervous system is a complex network consisting of bilions of neurons and synapses, it is characterized by a plastic ability, allowing to remodel, repair and reorganize its functions, even in presence of faults.[1] Artificial neural networks (ANN)s are a generalized mathematical model, inspired by the biological equivalent, representing today a valid and attractive solution for both industry and academy. Image classification performed in safety-critical applications (i.e. radars, flight control, robots), is the field where ANNs are used the most. [2]
A safety-critical system or a life-critical system is a system which, if subject to a malfunction or to a failure, may cause serious consequences to the environment or to users.[3] This premise highlights the urgent need for ANN to ensure high reliability and tolerance to faults.[4] Being the biological model counterpart, artificial neural networks (ANN) are considered inherently robust and fault tolerant.[5] This assumption derives in particular from two characteristics of the model [6]:

- over-provisioning: in a generic network, the number of neurons is higher than the minimum amount of neurons required to provide an output prediction. [7]
- a parallel and distributed structure.

The Figure 1.1 reported below shows the structure of a generic neural network. Thanks to the characteristics described, a network is able to produce the expected output even if a subset of its neurons are subject to *behavioural errors* (i.e. a neuron stops its activity or sends a different value than the expected one). However, when the number of errors exceeds the acceptable threshold, network reliability and accuracy degrades quickly.

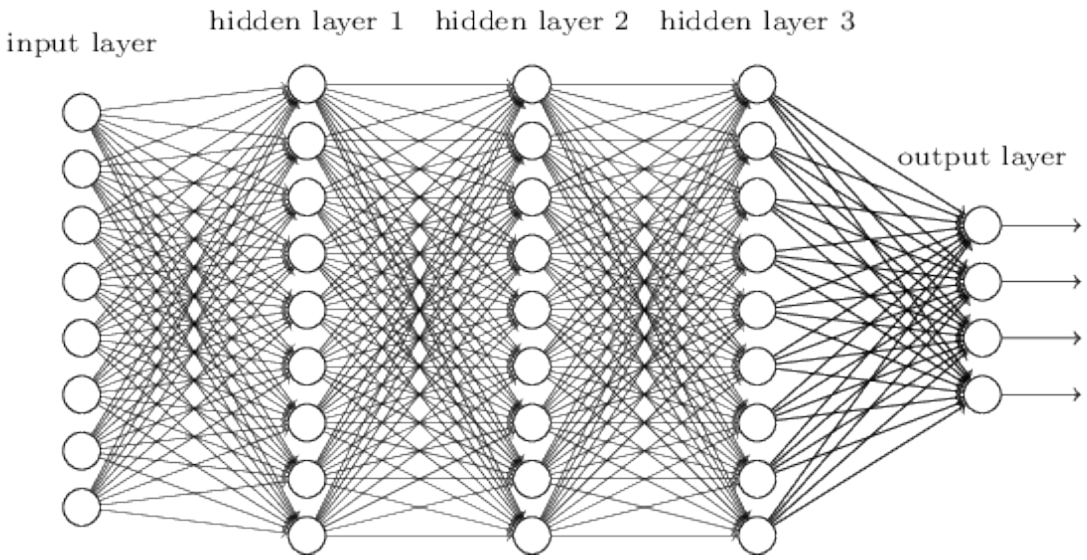


Figure 1.1: Common ANN architecture.

Motivated by the above-mentioned considerations, the objective of this study is to directly access the weak points of the network (i.e. all the neurons that most influence its robustness in case of a malfunction) and therefore, to propose a methodology that is able to overcome this scenario ensuring, even in case of behavioral errors, an high reliability. The key points of this research can be summarised in the following:

1. The first part of the methodology aims to identify which neurons are more critical within a neural network and to produce an ordered list of those neurons that are most influential in generating the final prediction. In particular, this technique involves two different levels of analysis: first, each neuron is considered as a processing element of each output class (class-oriented analysis); second, each neuron is seen as a processing element of the entire neural network (network-oriented analysis). Using a Residual Neural Network (ResNet-18) trained on two different datasets: MNIST and CIFAR-10, the proposed approach is validated by means of software fault injection (FI) campaigns.

2. Based on the criticalities identified in the previous step, to achieve network reliability, even in the presence of failures, the TMR mitigation technique was adopted. A fixed percentage of neurons were protected from individual behavioral errors and using this redundancy technique, the network is able to mask and thus to not propagate faults, within the successive layers of the network. At the end of this network fault-tolerance re-design, software FI

campaigns were again performed for validation purposes.

Will then be demonstrated that by using the proposed mitigation technique to protect critical neurons, even in the scenario where the network is subject to a high number of multiple faults, the result in terms of reliability is still high. It is necessary to specify that all the proposed considerations and approaches within the following chapters, result from an analysis of neural networks carried out exclusively at software level.
Chapter 2 provides background knowledge on ANNs, then the selected ResNet-18 is described and introduces the concepts of fault, error and failures and the main fault models in the field of artificial neural networks. The proposed approach is explained in details in Chapter 3. In Chapter 4, all the details about the implementation of the network and the tools used to develop it, are given. Next, Chapter 5 describes the experimental setup while Chapter 6 shows the experimental results. Finally Chapter 7 draws some conclusions and feature directions.

Chapter 2

Background

This chapter introduces background knowledge on the topics dealt during this Master Thesis. Initially, Section 2.1 introduces the biological and artificial model of neurons (2.1.1), presenting an overview on the neuron connection architectures (2.1.2), on ANN activation functions (2.2) and finally on Learning algorithms (2.3). Then, the main performance and architecture features of the selected residual neural network (ResNet-18) are described in Section 2.4. Finally, Section 2.4.4 provides an overview of the software-level fault models in DNNs.

2.1 Neural Networks

2.1.1 Models of Biological and Artificial Neurons

The human brain represents the pivotal example of the existence of massive neural networks capable of solving and managing in parallel cognitive and control tasks (e.g. face recognition, speech, body functions). Starting from the generalization of mathematical models of biological nervous systems and the introduction of a simplified model of neurons realized by McCulloch and Pitts (1943), the concept of ANNs has been developed.[8] The human brain consists of more than 10 bilion interconnected neurons and each neuron is shaped by a cell body or *soma*, by a single fiber, the *axon*, and several nerve fibers called *deindrites* (Fig.2.1a)[9]. Biologically, neurons transmit signals through synapses, a complex chemical process through which specific transmitters substances are released. Whenever the potential within the neuron cell has been stimulated enough by the incoming signals to reach the threshold, a pulse is sent down the axon (Fig.2.1c). However, if the required threshold level is not reached, no actions are generated. Analogously, an ANN consists of an elevated numbers of basic processing elements, called **neurons**, wired together by **weighted connections**. Comparing the models, biological

synapses becomes the connections weights that menage the input signals, while, the activation function is the artificial counterpart representing the non-linear behavior of biological neurons(Fig.2.1b). Referring to Figure 2.1b, the neuron's output signal y_j is given by the following relationship:

$$y_j = f(x, w) = f(\sum_{i=1}^{n}(x_i \cdot w_i)) \tag{2.1}$$

Where $x = x_1, x_2, ..., x_n$ are the input signals, $w = w_1, w_2, ..., w_n$ is the weight vector and the function $f(x, w)$ is the activation function.
The output neuron y_j is computed as:

$$y_j = \begin{cases} 1 \text{ if } (\sum_{i=1}^{n}(x_i \cdot w_i)) \geq \theta \\ 0 \text{ otherwise} \end{cases} \tag{2.2}$$

Where θ is the threshold level.
A multilayered neural network is illustrated in Figure 2.1d.

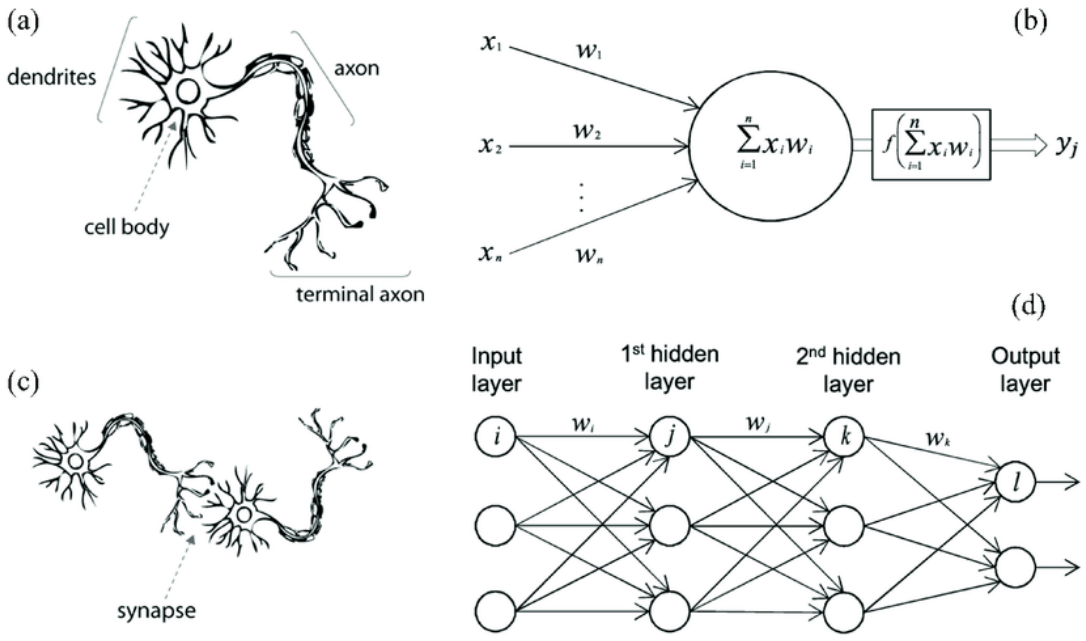


Figure 2.1: A biological neuron in comparison to an artificial neural network: (a) human neuron; (b) artificial neuron; (c) biological synapse; and (d) ANN synapses.

2.1.2 Artificial Neural Networks Architectures

The standard architecture of an ANN consists basically of three types of neuron layers: input, hidden and output layers. As for the connections between the different layers, one main group of networks is *feed-forward* .[8] In a feed-forward (Fig.2.2) neural network the information flows in only one course from the input nodes to the output nodes, (i.e., the output of any layer does not affect that same layer without creating loops). This type of architecture is also called bottom-up or top-down organization.[10]

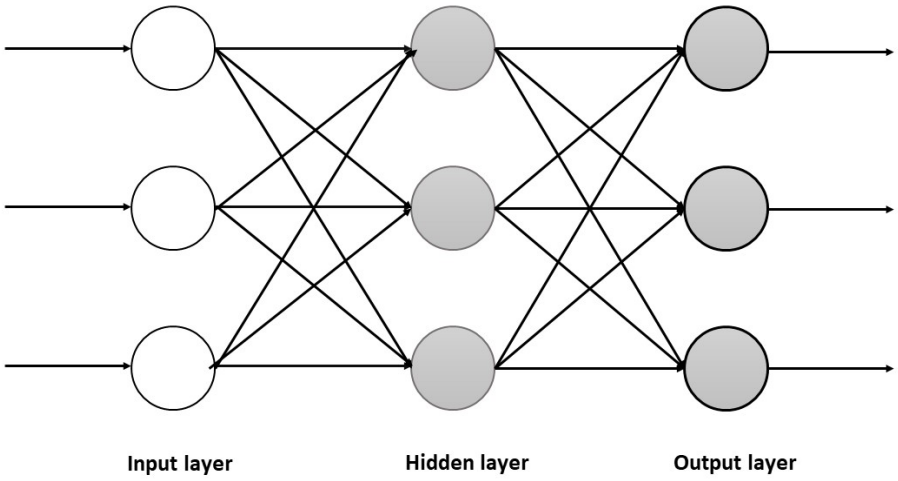


Figure 2.2: Feed-forward neural network

2.2 Activation Functions

The key role of the Activation Function in ANN is to decide whether a neuron should be activated or not. By calculating the weighted sum of input signals and by adding the bias, it usually limits the output range. [11] Activation Functions, also known as Transfer Functions, are distinguished mainly in linear and non-linear activation functions.

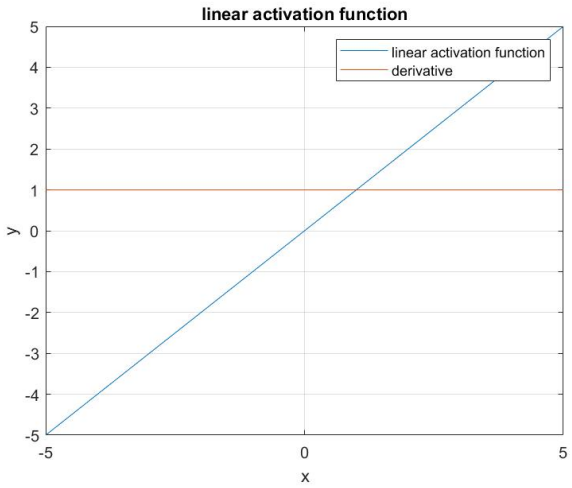


Figure 2.3: Linear Activation function.

2.2.1 Linear activation function

Linear activation function is also called *identity* or *no activation*.
As shown in the figure on top Fig. 2.3, the activation is proportional to the input following the relationship:

$$f(x_i) = kx_i \tag{2.3}$$

where x_i is the input of the activation function f and k is a fixed constant. The output of an identity function is k times the input, therefore the output's range can be $[-inf,+inf]$. Moreover, since the derivative is a constant, also the gradient is a constant and therefore, it is unrelated with the input. The main drawback of linear activation functions lies in the fact that the network cannot improve the error so that it is not able to identify complex patterns from the data.

2.2.2 Non-Linear Activation Functions

Non-linear activation functions are very used because they allow to interpret and process a wide variety of data and to range outputs differently. In this sub-sequence the three most commonly used non-linear activation functions will be introduced.

1. Sigmoid (S shaped) function.

   The Sigmoid Function is a nonlinear curve on the ranging [0,1] and is given by:

$$f(x_i) = \frac{1}{1 + e^{-x_i}} \tag{2.4}$$

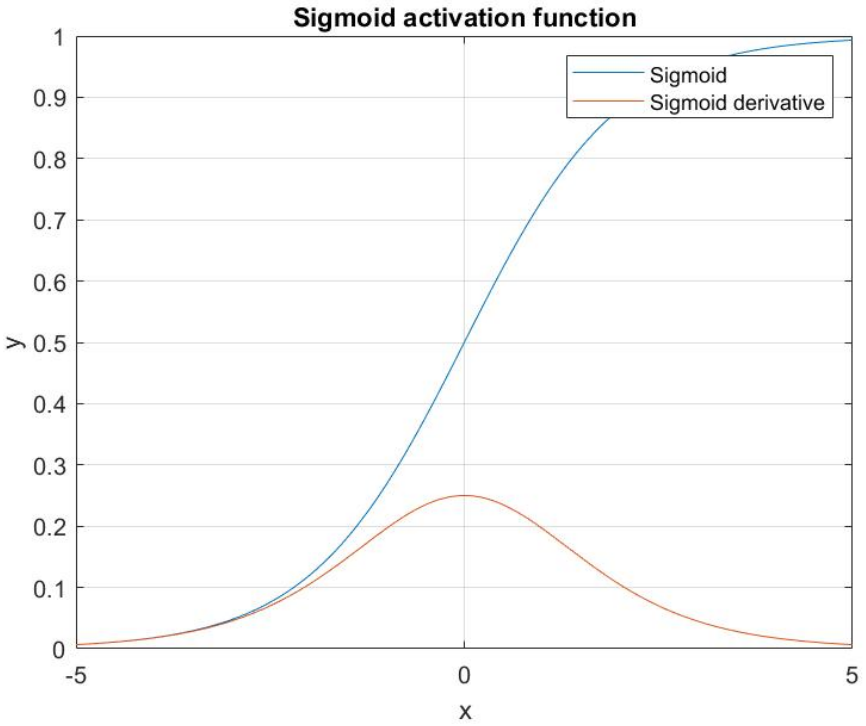


Figure 2.4: Sigmoid Activation function.

$$f'(x_i) = \frac{e^{-x_i}}{(1+e^{x_i})^2} \tag{2.5}$$

As shown in 2.4, the output of Sigmoid Function is steep, around 0 small changes of input would bring large changes of output, instead at both ends of the curve, the output tends to 0 or 1.

2. Tangent hyperbolic function

Tanh activation function represents a Sigmoid variant in which the output is zero-centered and is given by:

$$tanh(x_i) = \frac{2}{1+e^{-2x_i}} - 1 \tag{2.6}$$

$$tanh'(x_i) = \frac{4e^{-2x_i}}{(1+e^{-2x_i})^2} \tag{2.7}$$

Compared with Sigmoid Function, the key difference is the Tanh output ranging that is [−1,+1]. Although the curve is less steep than the Sigmoidal

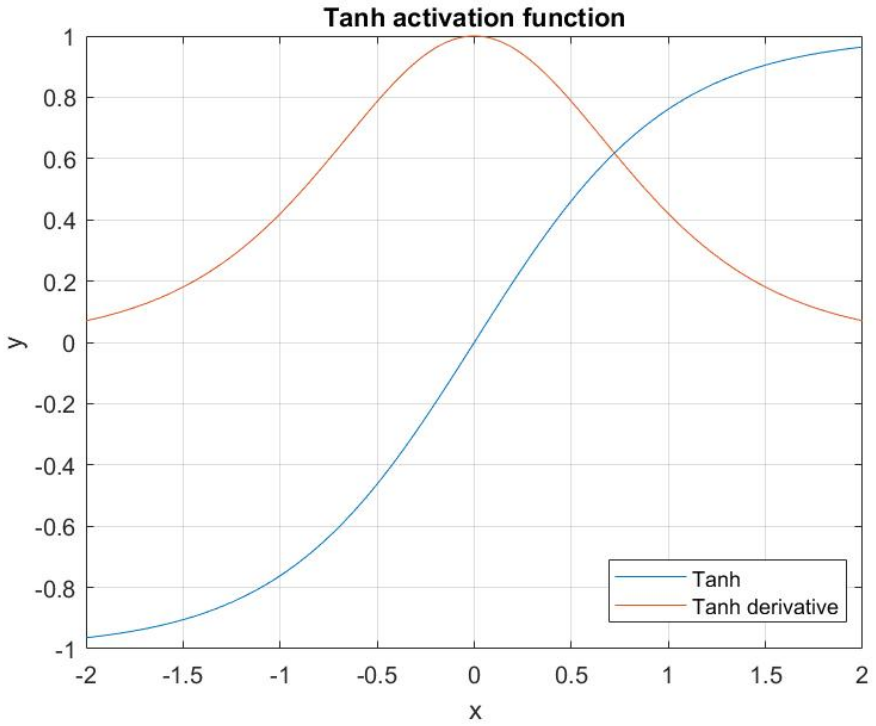


Figure 2.5: Tanh Activation function.

activation function, also in this case we find the problem of 'vanishing gradients'. Tanh activation functions are typically used in hidden layers of ANNs.

3. Rectified Linear Unit.

   ReLu is one of the most widely used activation functions especially when using Convolutional Neural Networks (CNN) or Deep Learning (DP).
   The main advantage of the ReLu is the fact that this activation function addresses the *vanishing gradient* problem. When we train a very deep neural network, using the activation functions described above, the gradient decreases exponentially propagating from the last layer to the first. Due to this behavior, the initial layers, often crucial for final prediction, are not actually updated in the training process. The ReLu, having constant and higher gradients than Sigmoid and Tanh, directly tackles the *vanishing gradient* problem. The ReLu activation function is given by:

$$f(x_i) = \begin{cases} x_i \text{ if } x_i \geq 0 \\ 0 \text{ if } x_i < 0 \end{cases} \tag{2.8}$$

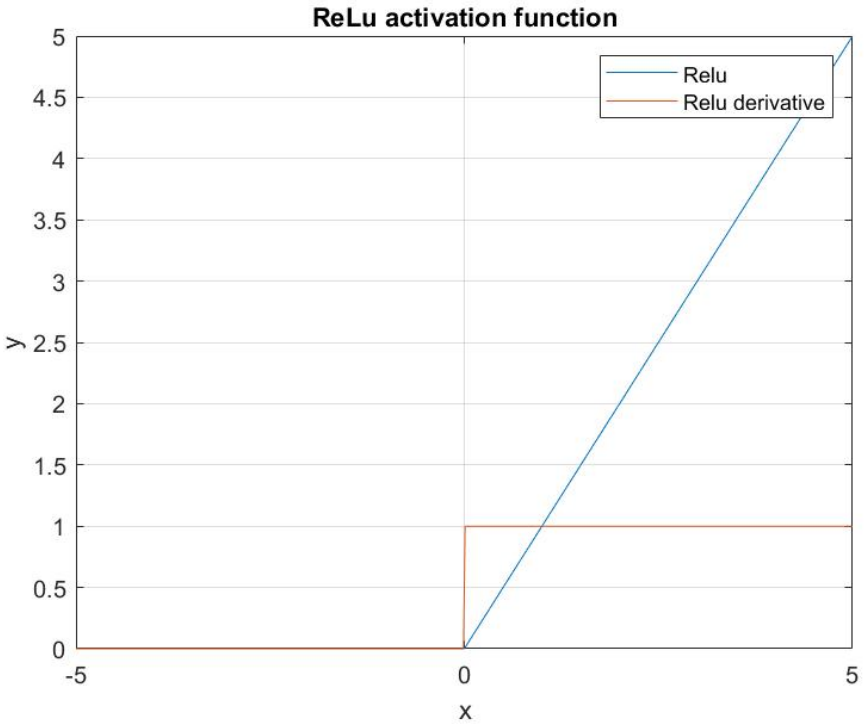


Figure 2.6: ReLu Activation function.

2.3 ANN learning process

Being ANN an adaptive complex system, the learning process implies that each processing element is capable of changing its input/output behavior due to the changes in environment. Basically, during the learning process the network adjusts its own weights to obtain better results in terms of accuracy.[12]
One of the main learning paradigms for ANNs is the **Supervised Learning**.
The Supervised Learning or *Associative learning* in which a sort of 'teacher' is defined which is able to pilot the training phase by providing pairs of input and output, the network learns prediction. In the field of ANNs such technique is used in feedforward models. Learning during the training phase, in supervised artificial neural network models, is also called *error back-propagation algorithm*, in which the network, based on the difference between the calculated output and the expected one, adjust the weights from the output layer until the input layer. The flow going forward from the input to the output and viceversa is called *epoch*. This recursive process of computations is continuously. till the network converges. [13]

2.4 ResNet-18

This section is entirely dedicated to the description of the neural network used throughout the entire project (i.e ResNet-18). Starting from some basic notions about Convolutional Neural Networks (CNN) 2.4.1, Section 2.4.2 describes the intuition behind the emergence of residual neural networks and the improvements to DP in the field of *Computer Vision*. Finally, in section 2.4.3, the main features of the ResNet-18 architecture are presented.

2.4.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) play a key role in image classification and recognition, in Computer Vision domain. An image (i.e a matrix of pixels) to be recognized is provided to the input layer of the network and its output is the predicted class label. A neuron in a later layer is related, unlike the networks described in the previous sections, only with some neurons in the preceding layer. This type of circumstantial correlation to a region is called a *receptive field* which, from a mathematical point of view, is represented by a matrix of pixels that are convoluted with the weight matrix, called **Kernel** matrix. The Kernel matrix is shared with all the receptive fields of the same input feature map, reducing decidely the number of trainable parameters. With this approach all relevant features, occurring at different locations in the input feature map can be detected.[14]
N different kernel applications, also named *convolution operations*, are able to

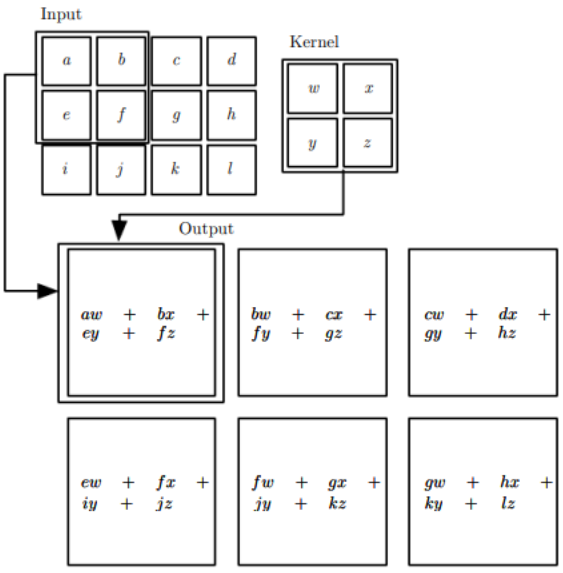


Figure 2.7: 2d Convolution operation.

extracts N number of features from the input image in a single layer representing distinct features, leading to N output feature maps. The kernel matrix is able to sliding horizontally as well as vertically, over the input image. As shown in the Figure 2.7, a convolution operation is nothing more than a sum of products element by element. In a CNN, in addition to the convolution layer is also defined a *Pooling layer*. The main benefits of this layer are first to reduce the dimensions of the feature maps as well as to reduces the number of learnable parameters, than to make the network much more robust to variations in the position of the features in the input image.

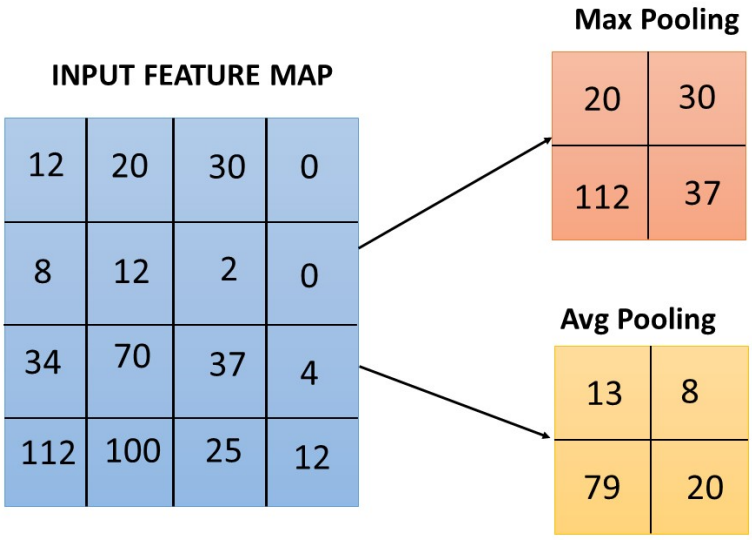


Figure 2.8: Pooling layer, a practical example.

Pooling layers can be *Max Pooling* or *Average Pooling*. While Max Pooling returns the maximum value from the portion of the image covered by the Kernel, the Average Pooling returns the average of all the values from the portion of the image covered by the Filter (Fig. 2.8). Finally, to produce the final prediction, the output of the previously described layers is passed to a *Fully Connected Layer*.

2.4.2 Residual Neural Network, intuition and improvements

Residual neural networks ResNet were first introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper "Deep Residual Learning for Image Recognition". [15]
Before the development of ResNet, to solve complex problems, additional layers were added creating deeper and deeper neural networks. However, it has been shown that the error in prediction increases with the depth of the network.[15] Hence the need to establish a depth threshold for CNNs. As shown in the Figure 2.9 in both cases of training and testing data, the error for 56-layer is more than a 20-layer network.

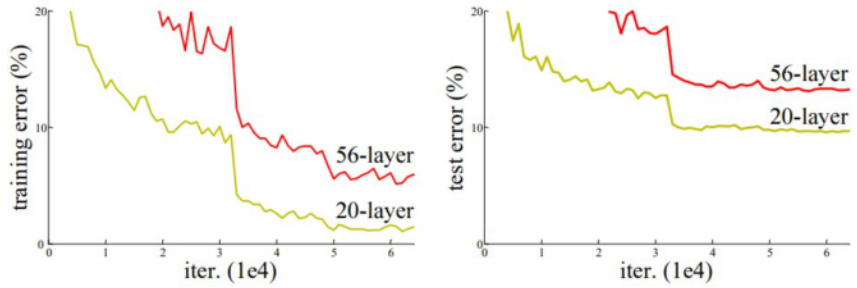


Figure 2.9: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks.[15]

When deeper neural networks begin to converge, degradation may occurs which leads to a clear decrease in network accuracy. This problem has therefore been dealt with by the introduction of ResNet, and in particular by the development of the so-called **Residual Blocks**.

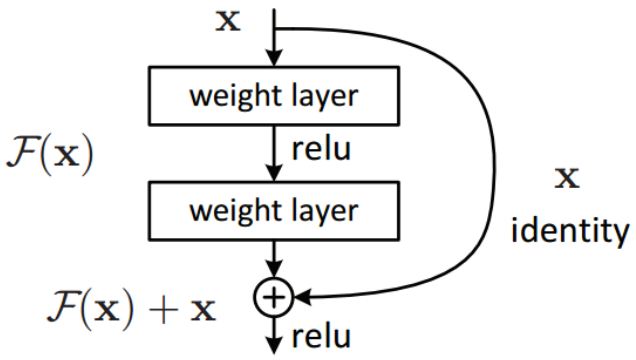


Figure 2.10: ResNet Skip connection. [15]

The residual blocks introduce a direct connection, which is called **skip connection**, which allows to skip some layers in the middle. The original mapping becomes now: $F(x) + x$ where x is the input feature map, therefore this architecture is also called *Identity mapping*. The use of residual blocks directly addresses the problem of vanishing gradient in DNN, allowing even the model to learn the identity functions. It is essential to stress that the use of skip connection does not increase the number of parameters and does not even affect the computational complexity of the network.

2.4.3 ResNet-18 architecture

The residual neural network (ResNet-18) used in this study consists of 18 layers.

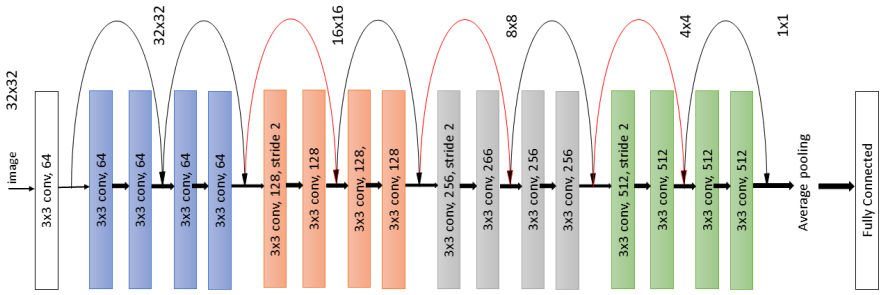


Figure 2.11: ResNet-18 Architecture

As shown in Figure 2.11, for each pair of convolution with 3x3 Kernel, there is a skip connection. This kind of organization is also called **Basic Block** which is a 2 layer deep Residual Block (Fig.2.12). Each Block takes as input a feature map characterized by a certain number of input channels and, after passing trough a flow of operations (i.e two Convolutional layers,Batch Normalization (BatchNorm),ReLu activation function), an output feature is obtained which is added to the input feature map. If there is a mismatch between the size of the feature maps that need to be added, the incoming feature map passes directly through the identity connection.
The use of BatchNorm inside the Basic Block, is another key step to gain improvements in network performances. BatchNorm is able to make fundamental improvements to the optimization problem, helping to tackle the *vanishing gradient*

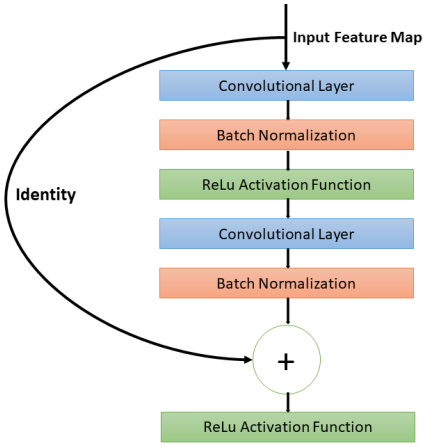


Figure 2.12: ResNet-18 Basic Block

problem and to obtain a faster network convergence. [16]

2.4.4 Software-level Fault Models in Deep Neural Networks

In the current section the concept of reliability of neural networks is introduced, emphasizing the distinction between faults, errors and failure. In particular, the issue of software faults for DNN will be addressed.
Nowadays, ANNs are a valid solution for applications that are considered safety-critical such as robots, flight control, space applications and self-driving cars. It is therefore necessary to ensure that these networks are considered reliable. It is a fact that neural networks are inherently reliable because on the one hand, they have a distributed and parallel structure and on the other, they are equipped with a number of neurons that is greater than the minimum number required to carry out the computational process.[17] Considering the biological parallelism, the human brain itself is able to tolerate that a small number of synapses or neurons do not behave as desired. Similarly, ANNs are able to menage the fault of a limited number of neurons due to over-provisioning. After providing this introductory scenario, the concepts of fault, error and failure are explained.[18]

- A *fault* or a *defect* is an anomalous physical condition in the system and corresponding to its temporal characteristic could be classified as permanent or transient. A permanent fault is an unrecoverable defect in the system, stable and fixed over time. A transient fault is instead, a defect active for a short period of time. A fault could gives rise to an error.

- An *error* is the manifestation of a fault in the system, for which the logical state of an element differs from its expected value. In the neural network scenario, each neuron is considered an independent identity that can fail regardless of failure of any other. Referring to behavioral error, we can distinguish between two categories of errors:
  - Crash: Neurons and synapses (i.e. the connections between different neurons) completely stops their activity. A crashed neuron can be modeled by using the dropout fault model, so that the neuron output is set to 0. At the other hand, a crashed synapses can be modeled as a synapses weighted by value 0.
  - Byzantine : Neurons or synapses send a value different from their nominal expected output of the activation function within their capacity.[7]
- A *failure* which is the propagation of errors at the system level. A failure evolves therefore, into a wrongly prediction of the output.

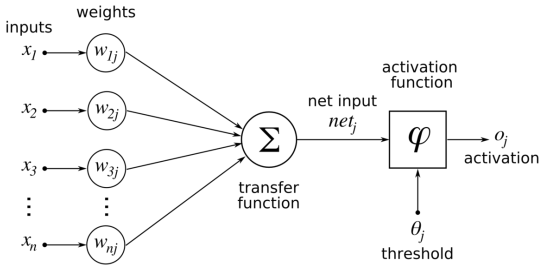
Chapter 3

Proposed Approach

The entire study aims to propose a methodology that is able to guarantee excellent results in terms of reliability of DNNs, even in the presence of behavioural errors at the behavioral level (i.e. crash neurons (2.4.4)). Going step by step, this chapter will describe the process through which neurons considered most critical for a neural network have been identified (3.1), and will explain in detail the redundancy method used to improve the accuracy of the network even in the presence of behavioral error.

3.1 Identification of the Criticality of Single Neurons

First of all when we refer to *critical neurons* we indicate the subset of neurons that contribute more to the final prediction of neural networks, in other words we refer to those neurons that carry more information than others. The study of artificial neural networks as mathematical models shows, as in the next figure, how the output of an artificial neuron is nothing more than the result of a summation, which then passes through the ReLu activation function. Being ReLu the activation function used, critical neurons are those that produce the highest value in output since they propagate their value within the network, contributing to the final prediction. Furthermore, by considering possible behavioural errors that may cause a change of sign in



[19]

the output, it is essential to consider the value of the neuron in absolute value. The theoretical-based criticality analysis is also reinforced by a further key phenomenon derived again from the biological counterpart. The memory of the human brain is distributed according to specific groups of neurons that are activated. In other words, depending on the input stimuli, the neurons are activated in particular pathways of neuronal activity. In the artificial neural model, this neuroscientific theory leads us to consider the net as a multi-output neural network, and therefore the contribution of a neuron can be seen from two different points of view. On the one hand, the neuron should be able to predict the output of the single class, on the other hand it should be able to provide the right prediction of the entire neural network. To clarify, considering a two-output neural network able to distinguish between dogs and cats pictures; following the explanation we can say that there are neurons more significant for the class of dogs and others for the class of cats. At the same time, however, all neurons must lead to a correct overall output prediction. On this premise is based the methodology of the ranking of critical neurons for an ANN. It consists of three steps:

- **Class-oriented analysis (CoA)**: For each individual output class during each training inference, the output values produced by each neuron are collected. At the end of the network training phase, the average in absolute value of the outputs of each neuron is calculated. Finally, according to the previous results a sorted decreasing list for each class is produced, so we can extrapolate which are the most critical neurons for each output class.

- **Network-oriented analysis (NoA)**: The process above is repeated for the entire neural network, without taking into account the distinctions between different output classes. A single sorted decreasing list is produced.

- **Final network-oriented list** : during this phase a single final list is created, considering both the criticalities per classes and the criticalities for the entire network.

It is important to point out that the search for critical neurons took place considering exclusively the convolutional layers, since the BatchNorm and the Activation Function (2.2), are only operations that modified the final output neuron, without adding other processing element to the network. Moreover, the output of each neuron is sampled after the BatchNorm and the Activation Function. Going into the detail of the methodology, during the CoA phase, the differentiated train dataset for each output classes is fed to the network and the output values of each neuron is therefore collected for every inference. At the end of the process, the average absolute value for each neuron is achieved. As for the NoA phase, the process of creating the ordered list remains unchanged but in this case the network is trained on the entire train dataset without any difference between the output classes. As explained

in the introduction to the chapter, a neural network is regarded as a multi-output neural network. Hence the need to carry out the third stage and generate the Final network-oriented list. It might happen that the critical neurons for individual classes take a low value in the NoA. The final sorted list of critical neurons is computed by executing the union without repetitions of all the classes' lists and the network list. All three steps mentioned above can be viewed in the image below, which is a practical example of the entire algorithm with CIFAR-10 dataset.3.1

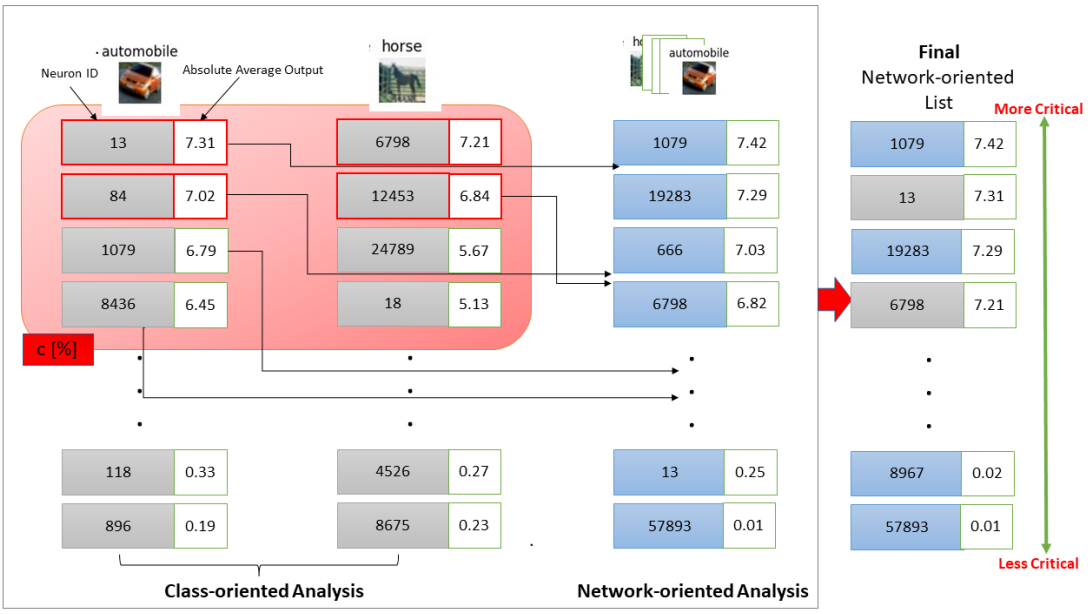


Figure 3.1: The critical neuron identification process : a pratical example with CIFAR-10 dataset

3.2 Triple Modular Redundancy on Artificial Neural Network.

As mentioned in the previous chapters, the entire study aims to propose a methodology that is able to ensure that an artificial neural network is able to guarantee a high accuracy, even in the presence of behavioural errors. The goal is therefore to create a neural network that can provide the right prediction even when one or more of its processing elements fail. In this study we considered that the neural network may be subject to a fault model whereby one or more neurons could be Crash or Byzantine neurons.(2.4.4) In this section is then presented the model of

fault tolerance design used to overcome this type of undesirable scenario.
First of all, among the various techniques of fault tolerance design at the software level, it is essential to introduce the concept of **redundancy**. Basically, a software system is redundant when it is able to perform the same functionality through the execution of different processing elements. In the field of redundancy techniques one of the most used approaches to meet the strict reliability requirements is the use of **Triple Modular Redundant** architecture. [5] The key feature of the TMR technique lies in the concept for which three separate units, working in parallel, performes the same process. The result is processed by a majority-voting element. Of course, it is necessary to specify right now that this type of architecture is able to mask the error of a single processing unit. In other words, if two of the three processing units are subject to a fault, the voter implemented in the TMR will not be able to mask the error. However, if we consider our specific scenario where neurons are supposed to be subject to behavioral errors, this kind of fault tolerance design, turns out to be an excellent compromise between the ability of getting a correct neural network's output prediction and memory footprint. In details, this modular redundancy technique is used to protect all those neurons that were identified as critic in the previous phase.

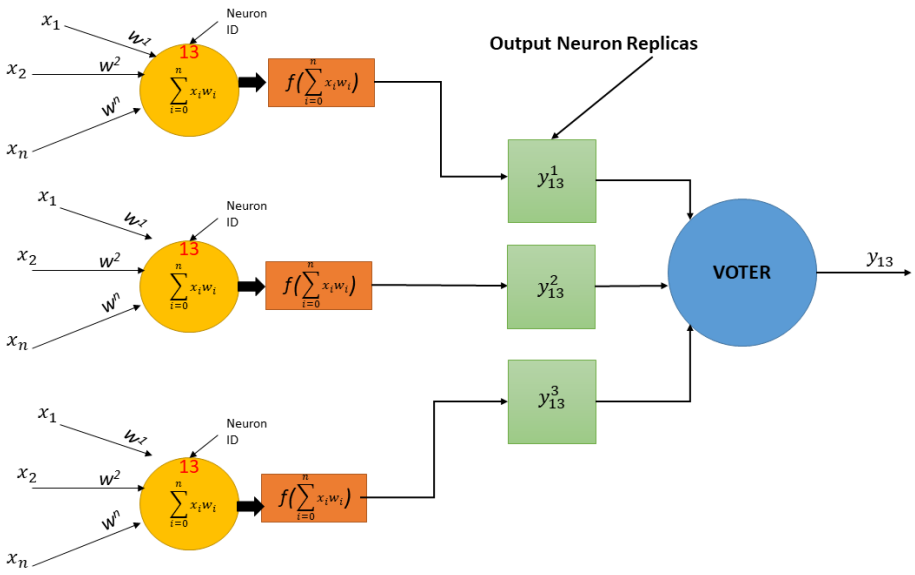


Figure 3.2: TMR implementation on a critical neuron

As shown in the picture above (Fig.3.2), the critical neuron identified with *ID13*, is duplicated within the network. In particular, each single replica performs in parallel, the same summation and then passes into its own activation function.

Each process element is therefore fed by the same duplicate input signals so as to obviate upstream errors due to writing and reading in memory of the weight matrices. Each neuron replicas produces its own output that feeds the majority voting unit. The voter in this case processes the outputs received by each processing modules and returns the majority output. Considering the particular case where two or all of the computation units are subject to behavioural errors (i.e. **multiple faults**), the voting unit is not able to return any majority. We have therefore agreed that in the event of the scenario mentioned above the output returned by the voter (i.e. y_{13}) is zero and that therefore the *neuron13* behaves like a Crash neuron without any possibility of masking the error.
Finally, as for the list of Crash neurons, extrapolated from the previous selection process, having been deemed not critical to the network, these are not protected but rather cut (i.e. their output is set to 0) so as to achieve greater network computational efficiency and better results in terms of memory footprint.

Chapter 4

Case study

In order to prove the effectiveness of the proposed methodology, the Residual Neural Network (ResNet-18) 2.4.3 is used, trained on two representative and famous dataset: CIFAR-10 [20] and MNIST [21].
The CIFAR-10 dataset (*Canadian Institute For Advanced Research*) is an object recognition dataset made of 60000 color images of size 32x32. It consists of ten output classes (i.e. *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck*) with 6000 images for each class. The CIFAR-10 dataset is composed of 50000 training images and 10000 test images and in particular the test split contains of exactly 1000 randomly-selected images from each class.[20](4.1)

Class	MNIST		CIFAR-10	
	Train	Test	Train	Test
1	5923	980	5000	1000
2	6742	1135	5000	1000
3	5958	1032	5000	1000
4	6131	1010	5000	1000
5	5842	982	5000	1000
6	5421	892	5000	1000
7	5918	958	5000	1000
8	6265	1028	5000	1000
9	5851	978	5000	1000
10	5949	1009	5000	1000
Total	60000	10000	50000	10000

Table 4.1: MNIST and CIFAR-10 Image Distribution

The MNIST database was created in 1998 as a combination of two of NIST's databases (i.e. *National Institute of Standards and Technology database*) : Special Database 1 and Special Database 3. In particular, Special Database 1 and Special Database 3 consist of digits written by high school students and employees of the United States Census Bureau, respectively.[20] The MNIST dataset (*Modified National Institute of Standards and Technology database*) is an handwritten digits recognition dataset that consists of 60000 training and 10000 test gray-scale images of size 28x28.As shown in the table 4.1, the distribution of MNIST images is not uniform for both the train set and the test set.
As regards the implementation of ResNet-18, it was developed using PyTorch [22] on a Windows server equipped with an Intel Core CPU i7-8565U and 8 GB of RAM. PyTorch is an optimized tensor library most used for DP and ML applications, in particular it is an open-source machine learning library for Python, mainly developed by the *Facebook AI Research team*, released in 2016. [22][23]. PyTorch is today one of the most used platforms both in industry and academia.

Layer Name	**Output Size**	**ResNet-18**
conv1	14x14x64	7x7,64 stride 2
conv2x	7x7x64	3x3 max pool, stride 2
		3x3,64, stride 1 3x3,64, stride 1 3x3,64, stride 1 3x3,64, stride 1
conv3x	4x4x128	3x3,128, stride 2 3x3,128, stride 1 3x3,128, stride 1 3x3,128, stride 1
conv4x	2x2x256	3x3,256, stride 2 3x3,256, stride 1 3x3,256, stride 1 3x3,256, stride 1
conv5x	1x1x512	3x3,512, stride 2 3x3,512, stride 1 3x3,512, stride 1 3x3,512, stride 1
fully connected	10	512x1000 fully connected

Table 4.2: ResNet-18 architecture trained on MNIST

ResNet-18 is a Residual Neual Network composed by 18 layer. In particular, as counted by the authors in the paper [15], the network is composed of a single initial convolution followed by 8 Residual Blocks 2.10, in which 2 convolutions are carried out and finally the fully connected layer FC. After each convolution it follows a BatchNorm and the ReLu activation function. ResNet-18, recives as input images of size 28x28x1 when trained on MNIST dataset, while 32x32x3 inputs when trained on CIFAR-10 dataset. For greater clarity, we can refer to the tables above and below to understand the ResNet-18 implementation customized on MNIST 4.2 and CIFAR-10 4.3.

Layer Name	**Output Size**	**ResNet-18**
conv1	32x32x16	3x3,16 stride 1
conv2x	32x32x16	3x3, 16 stride 1 3x3, 16, stride 1 3x3, 16, stride 1 3x3, 16, stride 1
conv3x	16x16x32	3x3,32, stride 2 3x3, 32, stride 1 3x3, 32, stride 1 3x3, 32, stride 1
conv4x	8x8x64	3x3,64, stride 2 3x3, 64, stride 1 3x3, 64, stride 1 3x3, 64, stride 1
Average Pooling	1x1x64	8x8 avg pool
fully connected	10	1x10 fully connected

Table 4.3: ResNet-18 architecture trained on CIFAR-10

Finally, ResNet-18 was trained and tested on MNIST dataset reaching a 95.53% of accuracy , instead trained and tested on CIFAR-10 dataset, it reaches an accuracy of 88.56%. Every detail of the implementation is then summarized in the table 4.4.

CNN Model	*dataset*	*Application*	*Accuracy*	*Total Neurons*	*Total Conv.Neurons*
ResNet-18	MNIST	Image Classification	94.53	43,072	39,424
	CIFAR-10	Object Recognition	88.56	131,136	131,072

Table 4.4: ResNet-18 benchmarks.

Chapter 5

Experimental setup analysis

During this chapter, the experimental setup is provided through an accurate discussion. In particular, each algorithm and the corresponding parameters used in the two macro-areas of the methodology will be presented gradually in two sections. In each of the two experimental setup paragraphs, the methodology adopted will be explained both as regards the *Identification of the Criticality of Single Neurons* (3.1) as regards the *Triple Modular Redundancy applied on critical neurons* (i.e.TMR technique) (3.2).

5.1 Critical Neurons Identification Experimental Setup

5.1.1 Class-Oriented Analysis (CoA) Setup

First of all, each training datasets has been declined into n subsets where n is the number of output Classes. For both datasets used, the number of output classes is 10. However, it is important to specify that this methodology can be suitable for any different number of outputs. Regarding MNIST dataset, each class represents an handwritten single digits between 0 and 9. CIFAR-10, instead contains 10 classes of images belonging to one of these categories: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The subsets of the training- datasets used in this phase, therefore, contain only images that represent the selected output class. For each training image of the selected output class, the output value assumed by each neuron has been saved in a list updated iteration by iteration. At the end of the collection process, the average was calculated, neuron by neuron, on each class list. Each of the 10 lists obtained by the algorithm is then ordered in descending manner: at the top we find the neurons activated with the highest average absolute value, at the bottom, instead those neurons less influential for the output class

under observation. In simpler words, the list for the particular output class, is sorted by neuron ID from the most critical to the least one.
Subsequently, software FI campaigns were performed in order to highlight that single output classes hold different robustness features in the presence of errors. At this stage, to inject behavioural errors to the neurons, we decided to adopt the Crash Neurons fault model. In particular, different percentages $p\%$ of neuron have been set to 0, deleting their contribution. To begin demonstrating the validity of the first area of the methodology (i.e. critical neurons identification), the same amount $p\%$ neurons outputs is set to 0 in two different scenarios.
In the first scenario (*Random*), the neurons to be switched off were chosen completely randomly from the class list.
In the second (*Critical*) scenario, the same number $p\%$ of neurons to be switched off is carefully selected from the top of the class list (i.e. selecting from the most critical neurons). After each round of FI, network accuracy was recalculated using the total number of test set images.
The experiments, independently from the dataset used, were developed for each output class of the target ResNet-18 and particularly, they were repeated for growing p-percentages: $p = 1.0\%$; $p = 10\%$; $p = 20\%$; $p = 30\%$; $p = 50\%$.

5.1.2 Network-oriented Analysis (NoA) Setup

In the previous section, software FI campaigns were performed to demonstrate that for each output class there is a set of neurons that are more relevant for the correct output class prediction. The same process of collecting neuron outputs is then reproduced at this stage, without differentiating by output classes. In detail, neuron outputs are collected for each image belonging to the entire training dataset and at the end of the process, for each neuron is calculated the average of the absolute values collected. The output of this process is now a single ordered list of critical neurons, where at the top we find the most critical neurons and at the bottom level, the neurons considered to be less important for the final prediction.

5.1.3 Final Network-Oriented list Setup

At the end of the two previous phases (CoA and NoA), a final network-oriented list is obtained on the basis of both the class and the whole train dataset criticalities. The approach used to obtain the final list is described in details in Section 3.1 (Figure 3.1). In particular, the final list represents the point of convergence of the two types of criticality that influence the performance of the entire neural network. By applying the explained algorithm, the merged list, without repetition, is generated. At this point, another software FI campaign is performed in order to

demonstrate that the final list takes into account both the output classes criticalities and the criticalities for the whole train dataset. Also in this case it was decided to compare two different scenarios: the case in which a set of neurons randomly chosen is crashed (*Random*) and the case in which a set of critical neurons is crashed (*Critic*). Again, the same fixed amount of neurons output (p) is set to zero and the resulting accuracy of the ResNet-18 was measured by running the total test set of images. Particulary, each experiment is replicated for growing p-percentages such as: p= 1%; p= 10%; p= 20%; p= 30% and p= 50%.

5.2 Triple Modular Redundancy Experimental Setup

The second section of this chapter is devoted to the setup of the TMR technique. This mitigation technique (3.2) is applied on a fixed percentage t of critical neurons which are duplicated within the network. To validate this fault tolerance design, a FI campaigns is carried out on growing p-percentage of randomly selected neurons in which a subset t of critical neurons is duplicated and then protected. In detail, firstly, the presented algorithm identifies within the percentage p of neurons to crash, a percentage t of the most critical neurons to which the TMR technique is applied. In other words, by considering the percentage p as a set of randomly

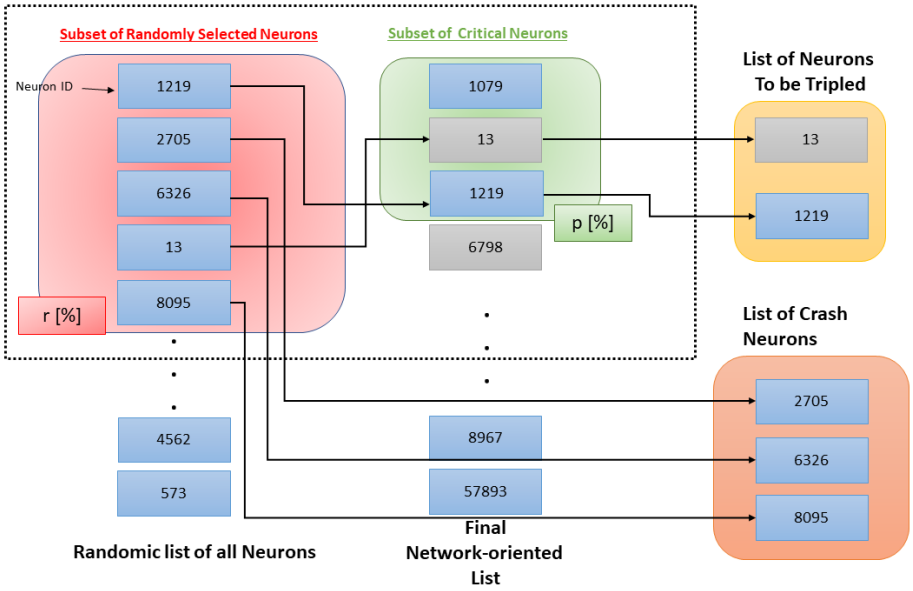


Figure 5.1: A Practical example of identifying the neurons to be tripled

selected neurons to be switched off and the percentage t as the set of the most critical neurons derived from the final network-oriented list, the algorithm returns on the one hand the difference set that is nothing more than the list of neurons that will actually be switched off during the campaign and on the other hand, generates the intersection set that is the list of critical neurons that the mitigation technique will duplicate. A practical example of how the algorithm works is reported in figure 5.1.

The direct advantage of using the TMR technique lies in the fact that even a critical neuron becomes a Crashed neuron, the other two replicas are able to process the expected output and then propagate it in the successive layers of the

network.(Fig.3.2) It is therefore a process of masking errors that is performing only when the failure acts on a single neuron among the three that the network has available to process that feature map input. Each experiment is replicated for growing p-percentages such as:
p= 1%; p= 10%; p= 30% and p= 50% while t is fixed for each FI campaigns and equal to 5%.

Chapter 6

Experimental result analysis

In this chapter, the experimental results are provided and, in particular, all the results obtained for the two macro-areas of the methodology will be presented gradually in subsections in which the outcomes of the ResNet-18 trained on MNIST dataset and ResNet-18 trained on CIFAR-10 dataset are respectively evaluated. In the each of the two experimental result analysis sub-paragraphs, the results obtained will be evaluated both as regards the *Critical Neurons Identification* (5.1) procedure as regards the *Triple Modular Redundancy* procedure (5.2) applied on critical neurons (i.e.TMR technique). All the experiments were performed using the open source machine learning framework PyTorch on a Windows server equipped with an Intel Core CPU i7-8565U and 8GB of RAM.

6.1 Critical Neuron Identification Experimental Results

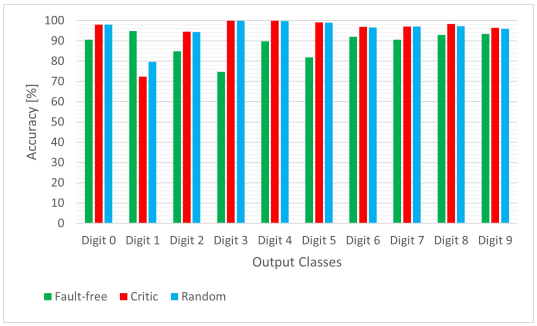
6.1.1 Class-Oriented Analysis (CoA) Experimental results

The experimental results of software FI campaigns on ResNet-18 trained on MNIST dataset are reported in the Figures 6.1(a,b,c,d,e) while, the outcomes on the CIFAR-10 dataset are shown in Figure 6.2(a,b,c,d,e).
The *Fault-Free* scenario is used as a reference parameter for the validation of the methodology and it is nothing more than the real accuracy of the class when the network is not subject to behavioural errors. It is important to underline that all the results shown have been obtained by running only the test inferences of the imagine belonging to the given output class.
By conducting a generalized analysis of both case studies, we can immediately validate the assumptions made in the previous chapters. On the one hand, it is evident that when the FI campaign is carried out on randomly selected neurons,

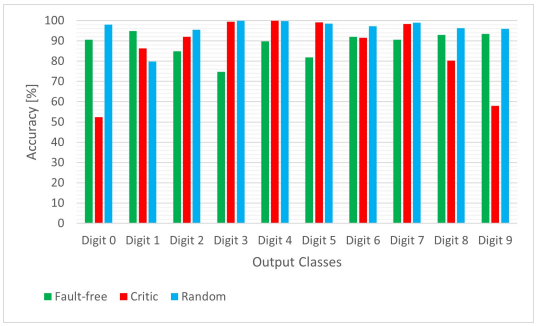
the accuracy of the network remains comparable with the *Fault Free* scenario. Excluding the case in which half of the neurons present in the network are switched off ($p = 50\%$), in the other 4 cases, the accuracy is comparable with the golden accuracy. From these experimental results, we can confirm that a neural network is equipped with a budget of neurons actually greater than the minimum number required to perform the right prediction.
On the other hand, it is clear that when the scenario shifts to the *Critical* one, the accuracy of the output classes decreases dramatically when those neurons, identified as critical, are killed. This last observation allows us to confirm that inside the network there are neurons that carry an higher number of information than the others and whether affect to behavioral errors, the performances of the entire network degrades quickly.
By looking at the graphs below, we can see that the results obtained with the CIFAR-10 dataset are better than those obtained with MNIST. It should be remembered, however, that the number of ResNet-18 neurons trained on CIFAR-10 is more than twice the number of neurons belonging the network when trained on MNIST dataset. It is therefore important to consider that in the second case, the budget of 'superfluous' neurons for prediction purposes is much higher and hence the need to interpret the results obtained in this perspective.

CoA on MNIST

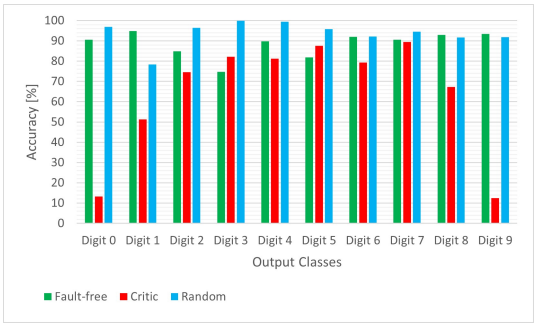
Concerning ResNet-18 trained on MNIST dataset, when $p = 10\%$ the accuracy of some output classes decreases dramatically. In particular, the first and tenth



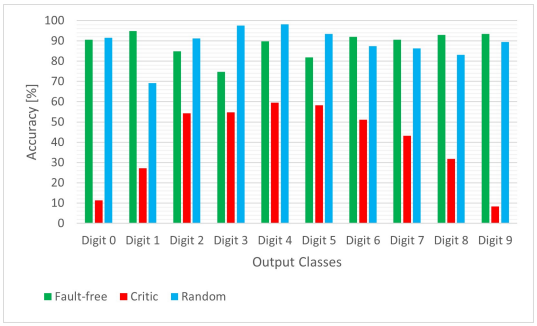
(a) $p = 1\%$



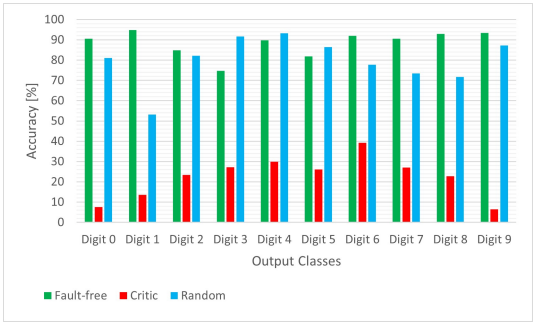
(b) $p = 10\%$



(c) $p = 20\%$



(d) $p = 30\%$



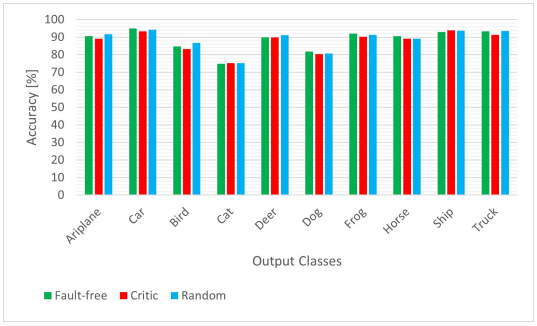
(e) $p = 50\%$

Figure 6.1: Class-Oriented analysis on MNIST

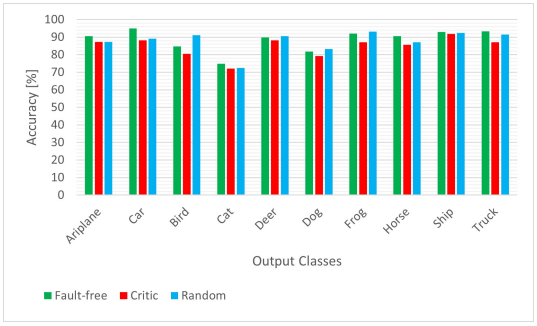
classes (which correspond to digital 1 and digital 9 respectively) show a variation in accuracy of 44.58% in the first case and 39.76% in the second case. This degradation in terms of reliability gets bigger and bigger when you go on to consider a higher

number of neurons killed. In detail, when 50% of the critical neurons of class 10 are switched off, the network accuracy on the subclass reaches 6.5%. The most robust class for the MNIST Dataset is instead, the class corresponding to digit 6. As for the case where the percentage p = 1%, we get improvements regarding accuracy. This behaviour might be attributed to a general over-fitting problem.

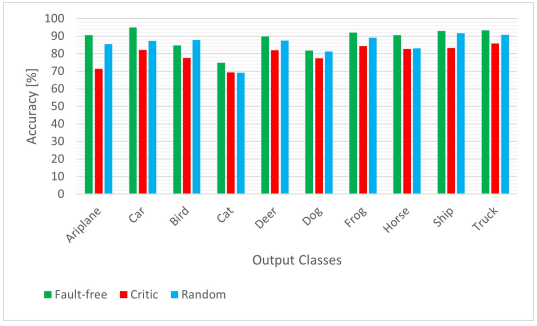
CoA on CIFAR-10



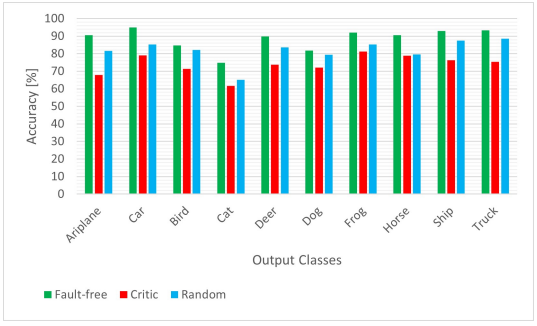
(a) $p = 1\%$



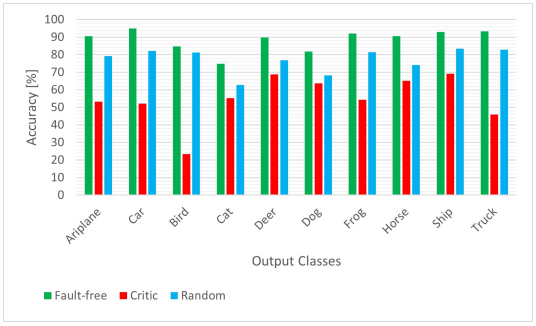
(b) $p = 10\%$



(c) $p = 20\%$



(d) $p = 30\%$



(e) $p = 50\%$

Figure 6.2: Class-Oriented analysis on CIFAR-10

Concernign ResNet-18 trained on CIFAR-10 dataset, when $p = 20\%$ the accuracy of some output classes decreases dramatically. Also in this case, due to over-fitting problems, when the network is lightened by killing some neurons, the accuracy of the network itself improves. However, we can see that when p = 30%, in the critical scenario, the accuracy of the network decreases significantly in each output class. The *'bird'* class is the one that is more subject to a negative variation in accuracy. In particular, it starts from a value of 84.8% up to a value of 23.43% when half of its critical neurons are killed. The variation is therefore 61.37%. From the results of the analysis, we can point out that the most robust class is the *'Ship'* class , immediately followed by the *'Deer'* class.

6.1.2 Final Network-Oriented list Experimental Results

The experimental results for the two software FI campaigns are reported in Figure 6.3 and Figure 6.4. In particular, in every graph we find on the x axis the percentage p of neurons killed and on the y axis the recalculated accuracy. The results of this analysis allow us on the one hand to strongly confirm what has already been learned from the previous section (i.e. in the event that critical neurons become Crash neurons for the network, accuracy decreases dramatically), and on the other hand, to validate the theory that for a NN should be considered both of the two criticality categories mentioned above.

Final Network-Oriented analysis on MNIST

Referring to the outcome of the ResNet-18 (Fig.6.3) FI campaign trained on MNIST dataset, it is immediately evident how the network is subject to an high degradation of its performance in terms of accuracy in the *Critical* scenario. In detail, when $p = 20\%$, the accuracy variation is 72.11%, reaches a value of 22,42%. For subsequent p samples, the degradation of the network remains rather constant, reaching a plateau value of 11.3% which is the accuracy of a*Random Classifier*. Referring instead to the blue curve, the *Random* scenario, we confirm again the theory that the network is equipped with a higher number of neurons with respect to the value strictly necessary. Is therefore highlighted a significant degradation in terms of accuracy only when p reaches 50%.

Final Network-Oriented analysis on CIFAR-10

Referring, instead, to the outcome of the ResNet-18 (Fig.6.4) FI campaign trained on CIFAR-10 dataset, we confirm that the network is equipped with a number of neurons much higher than those needed to make a correct prediction. This assumption is evident if we consider the fact that the *Random* scenario and the

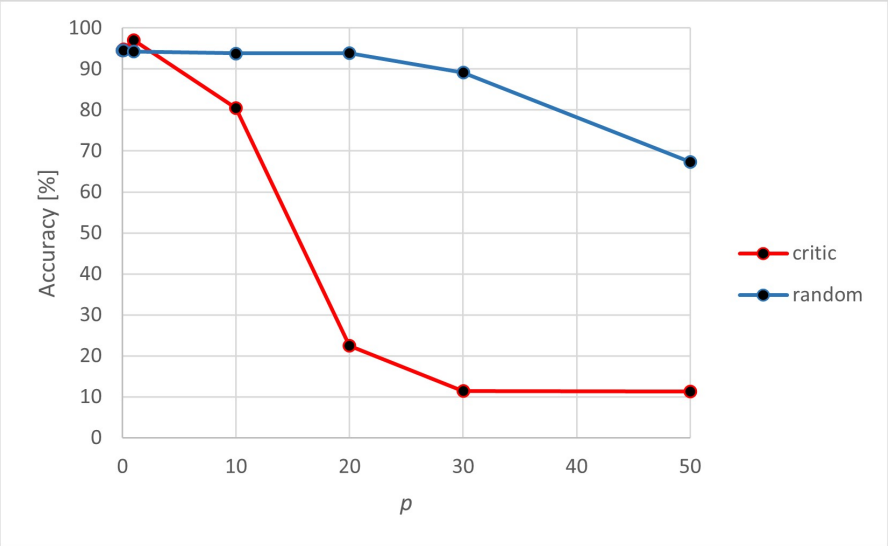


Figure 6.3: Final Network-Oriented Analysis on MNIST

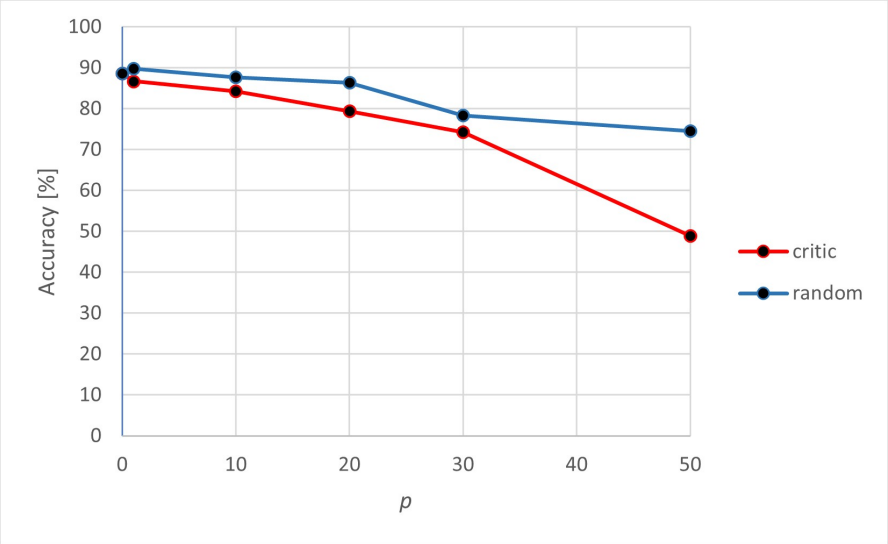


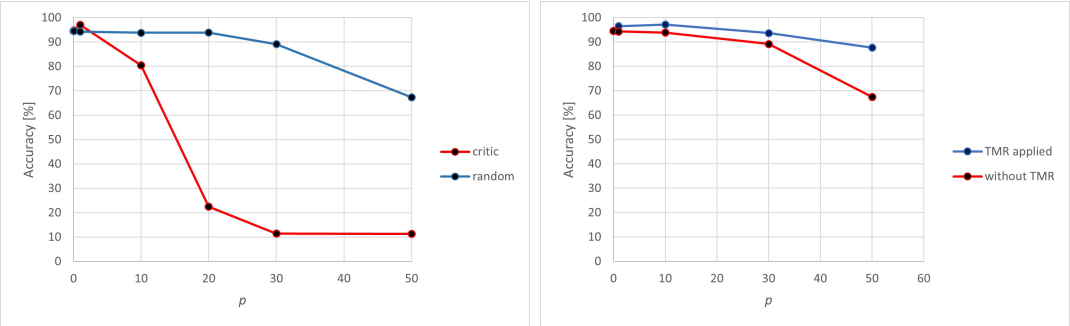
Figure 6.4: Final Network-Oriented Analysis on CIFAR-10

Critical scenario differ sharply only when p is more than 30%. In any case, following the behavior of the curve in red (*critical* scenario), we can note that the accuracy variation is 39,64% when p = 50% , while in the case of the (Random) scenario this variation is only equal to 14,11%. This type of comparison is strongly explanatory to validate the assumption that, in the event that critical neurons are faulty, network accuracy decreases rapidly.

6.2 Triple Modular Redundancy Experimental Results

The results obtained clearly show that the application of this redundancy technique is effective to ensure that the network correctly predicts the final output even when the injection of single faults affects critical neurons.
In order to make the result obtained clearer and more evident, the results of the analysis of the previous section are compared in the graphs below (Fig.6.5a, Fig.6.6a) when no percentage of critical neurons within$p$ were replicated) and the results obtained from the application of TMR on t of the critical neurons belonging to p (Fig.6.5b, Fig.6.6b). In every graph we find on the x axis the percentage p of neurons killed and on the y axis the recalculated accuracy. Each experiment is replicated for growing p-percentages such as:
p= 1%; p= 10%; p= 30% and p= 50% while t is fixed for each FI campaigns and equal to 5%.

TMR Analysis on MNIST



(a) Accuracy Evaluation of FI campaigns without TMR mitigation technique on MINST.

(b) Accuracy Evaluation of FI campaigns with TMR mitigation technique on MNIST.

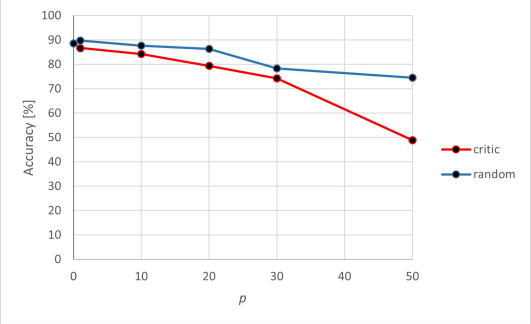
Figure 6.5

Referring to the results obtained from the application of the redundancy technique on the TMR dataset present in the graph above, we immediately note that the improvements in network accuracy due to the application of the mitigation technique are obvious and explanatory. By protecting 5 % of the critical neurons within p from time to time, the expected results are evident already for p= 10 %. In particular, whether in case in figure 6.5a, for p = 10 % the accuracy of the network was 80,51 %, after the application of the TMR, in Figure 6.5 b, the calculated accuracy is 97,17 %. To feed the effectiveness of the methodology, by

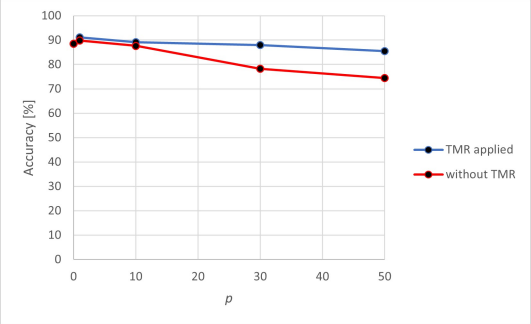
analyzing the sample $p = 30\%$, we note that in case a, the accuracy is 11.37 % while in case b is 93.65 %.

TMR Analysis on CIFAR-10

Referring to the results obtained from the application of the redundancy technique on the CIFAR-10 dataset shown in the graph below (Fig. 6.6), we further confirm the effective results obtained from the proposed methodology. Analyzing the experiments in detail, we notice a clear improvement of the response of the network to the faults injected at $p = 30\%$ where in case a, the accuracy is 74.21 while in case b is 87.91. Any doubt about the improvements brought by the use of the redundancy technique used, is resolved by looking at the results obtained for $p = 50\%$, where in the case a, the accuracy drops to 48.92%, while in case b is equal to 85.44% (a number very close to the initial golden value).



(a) Accuracy Evaluation of FI campaigns without TMR mitigation technique on CIFAR-10.



(b) Accuracy Evaluation of FI campaigns with TMR mitigation technique on CIFAR-10.

Figure 6.6

6.3 TMR Memory Footprint Drawback

We conclude this chapter reaffirming the success in the validation of the entire methodology and focusing briefly on the disadvantage in the use of this technique : the increasing of memory footprint due to the triplication of some process elements. If we focus in detail on the algorithm implemented by the methodology, we can notice that a variable number of neurons will be switched off within the network. This operation is therefore fundamental in the speech related to the occupied memory, as it allows us to cancel the computational effort and the occupation in memory of a significant number of processing elements.
Referring to ResNet-18 trained on MNIST dataset, the increase in memory is about 0.03%, while in the case of ResNet-18 trained on CIFAR-10 dataset. the increase is around 0.01%. We end the discussion by pointing out that all the experiments were carried out only at software level and that therefore these results represent an excellent trade off between memory footprint and network reliability in this specific context.

Chapter 7

Conclusions

This thesis provides a methodology able to improve the reliability of ANNs, showing how the most critical neurons can be identified and how they can be protected through a specific redundancy technique. Following the proposed methodology, has been demonstrated that ANNs are equipped with a number of processing elements (i.e. neurons), grater than the minimum value required to correctly perform the output prediction. The goal was therefore to understand which neurons were superfluous and which neurons were indispensable in terms of network reliability. During this study it was also explained how the criticalities of neurons should not be evaluated only for the entire network but also for the different output classes. In this regard the methodology is declined in two different stages of critical neurons identification : Class-Oriented Analysis (CoA) and Network-Oriented Analysis (NoA). These two phases converge in the third and last step of the first macroarea: Final Network-oriented analysis from which critical neurons are extrapolated. Based on the results obtained during the validation of the critical neuron identification algorithm, the methodology continues by proposing the use of a famous redundancy technique with the aim of making the network more robust, even in the presence of behavioural errors. Also in this case the validation phase was more than satisfactory and we have demonstrated, through the use of fault injection campaigns at the software level, that the network, even in the presence of single faults on neurons, maintains a high accuracy.
To conclude, in the feature, the objective is to validate this same methodology also considering the hardware level of the implementation.

Bibliography

[1] Terry Sejnowski and Tobi Delbruck. «The Language of the Brain». In: *Scientific American* 307 (Oct. 2012), pp. 54–9. DOI: 10.1038/scientificamerican1012-54 (cit. on p. 1).

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV] (cit. on p. 1).

[3] A Dictionary of Computing. . Encyclopedia.com. *"safety-critical system ."* Nov. 2021. URL: https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/safety-critical-system (cit. on p. 1).

[4] Annachiara Ruospo, Angelo Balaara, Alberto Bosio, and Ernesto Sánchez. «A Pipelined Multi-Level Fault Injector for Deep Neural Networks». In: *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)* (2020), pp. 1–6 (cit. on p. 1).

[5] Alberto Bosio, Paolo Bernardi, Annachiara Ruospo, and Ernesto Sanchez. «A Reliability Analysis of a Deep Neural Network». In: Mar. 2019, pp. 1–6. DOI: 10.1109/LATW.2019.8704548 (cit. on pp. 1, 20).

[6] Steve Lawrence, C. Giles, and Ah Tsoi. «What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation». In: (Mar. 2001) (cit. on p. 1).

[7] El Mahdi El Mhamdi and Rachid Guerraoui. «When Neurons Fail». In: *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (May 2017). DOI: 10.1109/ipdps.2017.66. URL: http://dx.doi.org/10.1109/IPDPS.2017.66 (cit. on pp. 1, 16).

[8] Ajith Abraham. «Artificial neural networks». In: *Handbook of measuring system design* (2005) (cit. on pp. 4, 6).

[9] Ali Zilouchian. «Fundamentals of neural networks». In: *Intelligent control systems using soft computing methodologies* 1 (2001), pp. 1–5 (cit. on p. 4).

[10] G. Bebis and M. Georgiopoulos. «Feed-forward neural networks». In: *IEEE Potentials* 13.4 (1994), pp. 27–31. DOI: 10.1109/45.329294 (cit. on p. 6).

[11] Jianli Feng and Shengnan Lu. «Performance analysis of various activation functions in artificial neural networks». In: *Journal of physics: conference series*. Vol. 1237. 2. IOP Publishing. 2019, p. 022030 (cit. on p. 6).

[12] Harsh Kukreja, N Bharath, CS Siddesh, and S Kuldeep. «An introduction to artificial neural network». In: *Int J Adv Res Innov Ideas Educ* 1 (2016), pp. 27–30 (cit. on p. 10).

[13] R. Sathya and Annamma Abraham. «Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification». In: *International Journal of Advanced Research in Artificial Intelligence* 2 (Feb. 2013). DOI: 10.14569/IJARAI.2013.020206 (cit. on p. 10).

[14] «Performance Analysis of Various Activation Functions in Artificial Neural Networks». In: *Journal of Physics: Conference Series* () (cit. on p. 11).

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (cit. on pp. 13, 24).

[16] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. *How Does Batch Normalization Help Optimization?* 2019. arXiv: 1805.11604 [stat.ML] (cit. on p. 15).

[17] Annachiara Ruospo and Ernesto Sanchez. «On the Reliability Assessment of Artificial Neural Networks Running on AI-Oriented MPSoCs». In: *Applied Sciences* 11.14 (2021) (cit. on p. 15).

[18] Cesar Torres-Huitzil and Bernard Girau. «Fault and error tolerance in neural networks: A review». In: *IEEE Access* 5 (2017), pp. 17322–17341 (cit. on p. 15).

[19] Wikimedia Commons. *File:ArtificialNeuronModel english.png — Wikimedia Commons, the free media repository*. [Online; accessed 25-November-2021]. 2021. URL: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:ArtificialNeuronModel_english.png&oldid=592436755%7D (cit. on p. 17).

[20] Alex Krizhevsky. «Learning Multiple Layers of Features from Tiny Images». In: (2009), pp. 32–33. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (cit. on pp. 22, 23).

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791 (cit. on p. 22).

[22] He Lyu, Ningyu Sha, Shuyang Qin, Ming Yan, Yuying Xie, and Rongrong Wang. «Advances in neural information processing systems». In: *Advances in neural information processing systems* 32 (2019) (cit. on p. 23).

[23] Nikhil Ketkar. «Introduction to PyTorch». In: *Deep Learning with Python: A Hands-on Introduction*. Berkeley, CA: Apress, 2017, pp. 195–208. URL: https://doi.org/10.1007/978-1-4842-2766-4_12 (cit. on p. 23).