ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA

DIPARTIMENTO di
INGEGNERIA DELL'ENERGIA ELETTRICA E DELL'INFORMAZIONE
"Guglielmo Marconi"
DEI

CORSO DI LAUREA MAGISTRALE IN
Advanced Automotive Electronic Engineering

TESI DI LAUREA
in
Test, Diagnosis and Reliability

Review of Fault Mitigation Approaches for Deep Neural Networks for Computer Vision in Autonomous Driving

CANDIDATO

Mattia Cerino

RELATORE

Chiar.ma Prof.ssa Cecilia Metra

CORRELATORI

Dr. Martin Eugenio Omana
Dr. Michael Paulitsch
Dr. Ralf Graefe

Anno Accademico
2019/2020

Sessione *I*
21/07/2020

To my family and my friends

Index

Index……..……3

Introduction…….……...……..……...……..5

Motivations……....…….……...…6

1. Chapter 1: Preliminary concepts……..…..……...…….…..7

1.1. Characterization of Autonomous Driving…….8

1.2. Review of typical Hardware Failures and Threats……9

1.3. Summary of the standard ISO 26262……..11

1.4. Introduction to Deep Neural Networks……15

1.5. Deep Neural Networks in Safety-Critical Systems…….19

2. Chapter 2: General Reliability Evaluation Framework…….……..20

2.1. Impact of General Errors on DNNs' Accuracy……...……21

2.2. A Case Study on Bit Transitions…….……23

2.3. Reliability Tests……..…….24

2.3.1. Neutron Beam Experiments……..25

2.3.2. Fault-Injection Framework……...……26

2.4. A Fault Vulnerability Evaluation Methodology……..……27

2.5. Error Criticality Evaluation……...……..30

3. Chapter 3: Permanent Faults Mitigation on Deep Neural Networks' Hardware Accelerators…….……..32

3.1. A Methodology for Designing Reliable AI Systems……...……33

3.2. Deep Neural Networks' Accelerators and Graphic Processing Units.…35

3.3 Error Resilience Analysis……41

3.4. Methodologies Reducing the Impact of Permanent Faults……..43

4. Chapter 4: Transient Faults Mitigation…..…….……..48

4.1. General Impact of Soft Errors on Deep Neural Networks ….…..……49

4.2. General Impact of Soft Errors in Hardware Accelerators….……..….51

4.3. Methodologies Reducing Impact of Transient Faults on DNNs' Accelerators……..52
4.4. Selective Hardening on FPGA……...…….55
4.5. Brief Observation on Aging……59
Conclusions……..……63
Acknowledgements……..……64
Abbreviations……..……..……65
Bibliography……...……..……67

Introduction

Recently, the development of technology has led to a widespread usage of Machine Learning (ML) algorithms in autonomous systems, in particular in safety-critical autonomous cyber-physical systems. One of the key beneficiaries is Autonomous Driving. Nowadays, Deep Neural Networks (DNNs) are the trending state-of-the-art ML algorithms for most of the Artificial Intelligence (AI) applications. Computer vision is crucial to guarantee autonomous control, and DNNs promise to perform tasks as object detection, classification and localization with particularly high correctness. Nevertheless, the automotive industry has always been a competitive market in terms of strict safety requirements for both hardware and software designers. One purpose of Autonomous Driving is to exclude the human role on vehicle control.

Therefore, it is essential to guarantee high reliability: developing a robust DNN algorithm represents only a single side of the problem. The other side is the safety and reliability of the underlying hardware architecture: DNNs have an high non-linear nature, hence their decision bounds are not precise. Thus, hardware-level faults represent a substantial problem for DNNs implementation as they might expose to risk the entire system safety by leading to unforeseen network decisions.

Motivation

The aim of this work is to identify and present challenges and risks related to the employment of DNNs in Computer Vision for Autonomous Driving. Nowadays one of the major technological challenges is to choose the right technology among the abundance that is available on the market.

Specifically, here it is collected a synopsis of the state-of-the-art architectures, techniques and methodologies adopted for building fault-tolerant hardware and ensuring robustness in DNNs-based Computer Vision applications for Autonomous Driving.

Chapter 1
Preliminary Concepts

In the first chapter of this thesis work there will be a brief description of the most common characterization of the autonomous driving, a review of the typical hardware failures and threats, a summary of the main standard (ISO 26262) that rules the reliability of Autonomous Driving, and an introduction to Deep Neural Networks and their application on safety-critical systems.

1.1. Characterization of Autonomous Driving

In the SAE International's standard J3016: Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems [1] six levels of driving automation are identified. This is done in order to provide a common taxonomy to make simpler communication and collaboration between technicians.

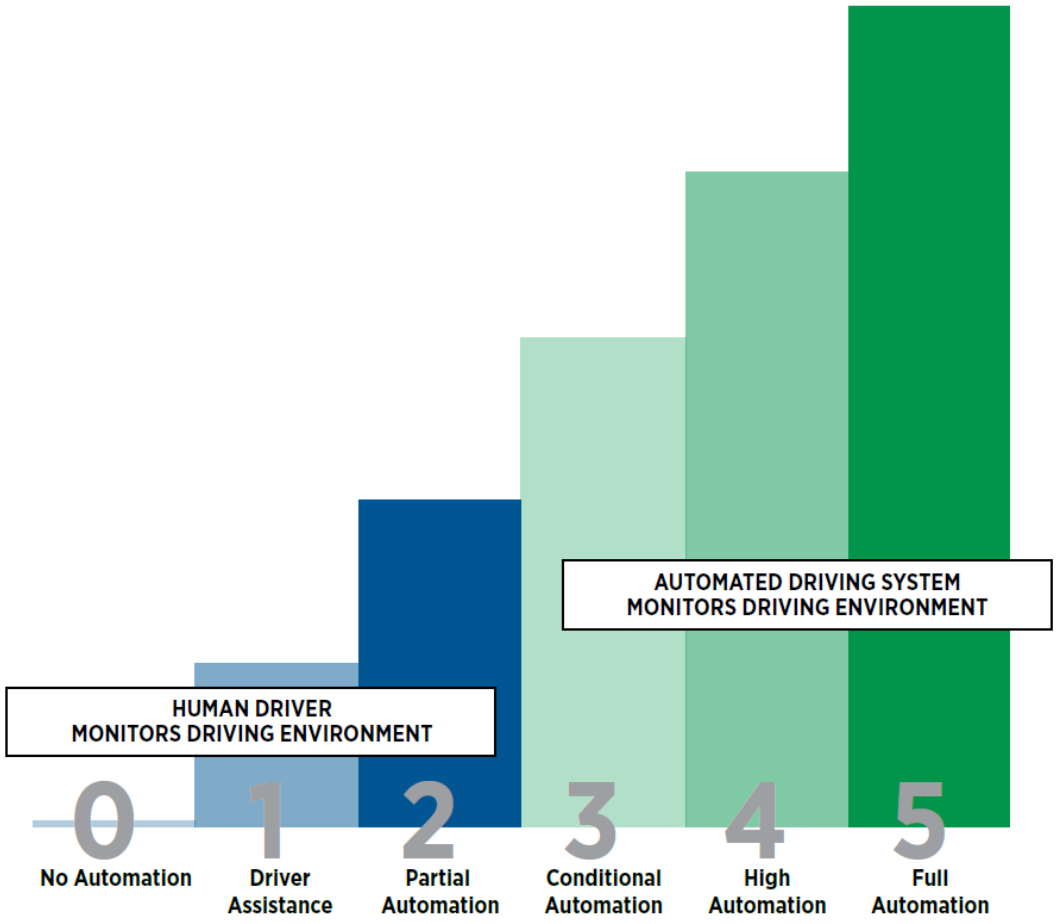

Figure 1.1. Levels of driving automation [1]

A key demarcation line can be drawn between level 2 and level 3: considering levels from 0 to 2 the human driver is in charge of monitoring the driving environment, while from level 3 to 5, it is the automated driving system the actual responsible of performing the entire dynamic driving task. It is worth noting that a specific vehicle under consideration can be provided by multiple

features, and depending on the number of features that are activated it could operate at different levels. These levels are summarized in a table, shown in figure 1.2:

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Figure 1.2. Description of levels of driving automation [1]

1.2. Review of typical Hardware Failures and Threats

In literature, the main idea behind dealing with failures, is not to target physical defects, but to classify them by the effect they have on the circuitry behaviour: this approach is known as finding and describing **Fault Models**. This strategy is crucial in testing and in diagnosis, but also in an approach called **Fault Tolerance**. The last one is referring to guarantee the correctness of operations of systems in the field, meaning that systems must be able to detect eventual faults and then to mask them or correct them as fast as possible. Depending on the target system, the consequences of a fault can be catastrophic or not. Autonomous Driving, which is the topic of this work, is a very important

example of safety critical application, with high dependability needs. **Dependability** is a global concept including Reliability, Availability, Safety and Security. The fundamental concept characterizing the Fault Tolerance approach is the fact that the design of products is reinvented in order to tolerate faults in the field: this is as successful as the designer is able to predict the kind of faults that can happen in the field. Therefore, it is very important to have available accurate and extensible fault models. Between traditional hardware fault tolerance techniques, it is possible to mention Modular Redundancy, Online Testing and Recovery and, specifically for memories, Error Correcting Codes.

Coming to the most common threats and hardware failures, they can be classified as hazards, potential sources of harm caused by malfunctioning behaviour of one circuit. It is worth noting the following ones:

- **Permanent faults**: faults that are always present from the moment in which they appear. Examples of these are transistor stuck-at, transistor stuck-on, transistor stuck-open, resistive bridging faults, crosstalk faults. Faults of this kind can be generated by Process Variations, meaning that there are imperfections in the chip fabrication process, and also by Aging, so degradation in circuit's characteristics over time.
- **Transient faults**: faults that are only temporarily present, then the system goes back to its correct operation. Faults of this kind are usually generated by external environmental events, like high energy particle strikes (cosmic rays producing energetic neutrons or alpha particles coming from package decay) or electromagnetic interferences. Also temperature of operation has a role in this context. The consequence of this class of faults are the so-called **Soft Errors**, that are basically transient bit flips.
- **Intermittent faults**: faults that are only sometimes present, an example are the ones due to external vibrations.

One of the key purpose of fault tolerance is to achieve a sufficiently high fault coverage with respect to the considered fault models, and to keep fault models considering a wide scope of possible failures.

1.3. Summary of the standard ISO 26262

Reliability study and its assessment is structured by different standards, released referring to different application domains.

ISO 26262: Functional Safety – Road Vehicles has been the main standard regarding Functional Safety in the automotive domain since 2011, when its first edition was published. Its purpose was to address specific safety requirements of Electrical and Electronic (E/E) systems embedded in road vehicles. It is a general standard: it is oriented to the whole safety lifecycle of systems' development. Indeed, functional safety features are part of each automotive product's development phase: specification, design, implementation, integration, verification, validation and product release. ISO 26262 does not address the nominal performance of E/E systems; it is about the possible hazards caused by malfunctioning behaviour of E/E safety-related systems. First, there is an identification of safety goals aiming to detect and moderate the effect of hazards, then as a consequence of these goals some requirements for the final architecture, in terms of both hardware and software, are listed. Nowadays it is applied the second version of this standard. In the standard functional safety is defined as "The absence of unreasonable risk due to any potential source of harm caused by malfunctioning behavior of electrical and or electronic systems" [2]. Then safety is instead defined as "The absence of unreasonable risk due to any potential source of harm caused by malfunctioning behavior of electrical and or electronic systems" [2]. In the standard there is also a portion (Part 2) regarding the Functional Safety Management, which discusses the activities to be completed during the safety lifecycle with annexed requirements. In the third part, there is a categorization of the potential hazards, that is done by the identification of three aspects: severity (importance of

consequences), probability of exposure (of the driver to the risky situation) and controllability (evaluated as action taken by a driver, it is the capability to avoid unsafe events). At the end of this, it is possible to do a classification, assigning to different hazards a certain level called Automotive Safety Integrity Level (ASIL). Everything is summarized in the table of figure 1.3. To properly read it, it is worth noting that the three aspects under investigation are evaluated as follows:

- Severity: between S1 (light and moderate injures) and S3 (fatal injuries);
- Probability of exposure: between E1 (very low probability) and E4 (high probability);
- Controllability: between C1 (simply controllable) and C3 (difficult to control or uncontrollable)

As a result, there is the definition of four ASILs:

- From ASIL A (the lowest safety integrity level) to ASIL D (the strictest safety integrity level).

It is possible to notice that there is also another label, QM, that stands for Quality Management, which is there to represent the cases in which there is no requirement in accordance with ISO 26262. Anyway, for other kind of requirements in fields of Durability, Quality, Reliability etc., those situations should be considered.

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 1.3. ASIL Determination [2]

The same typology of levels (ASILs) is assigned also to the safety goals, which are expressed to avoid or moderate the effect of an hazard. Finally, as a consequence, from safety goals, safety requirements are stated, and then they inherit the corresponding ASIL level.

The product development is directed by the Technical Safety Concept, that is locating the requirements in a precise way to the system components (hardware and software). When the final product is ready, to be compliant with the standard it is needed to pass a Safety Validation step, in which each requirement is verified. The whole process can be summarized in the following figure:

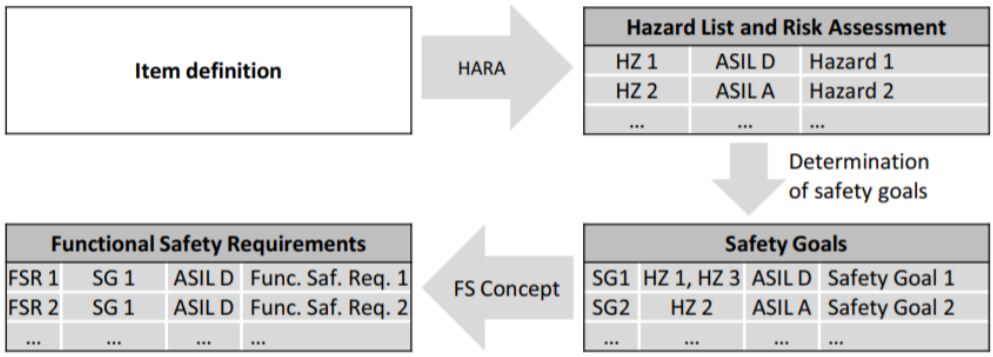

Figure 1.4. Flow of identifying hazards to creating functional safety requirements [3]

The portion of the standard which is of particular interest of this thesis work, is the one regarding the technical safety concepts in the hardware development. It is well summarized in the article by Jung *et al.* [4]. Every level of ASILs is characterized by its own procedures regarding items development. Technical specifications about hardware compliance are based on some metrics that relate an hardware element with respect to its failure modes. The analysis specified by the standard is referred to random hardware failures affecting E/E parts. The neglection of systematic failures is done supposing they are detected and treated during initial tests.

To define and evaluate the hardware architectural metrics, it is needed a list with precise definitions of failure modes. Figure 1.5 is summarizing them.

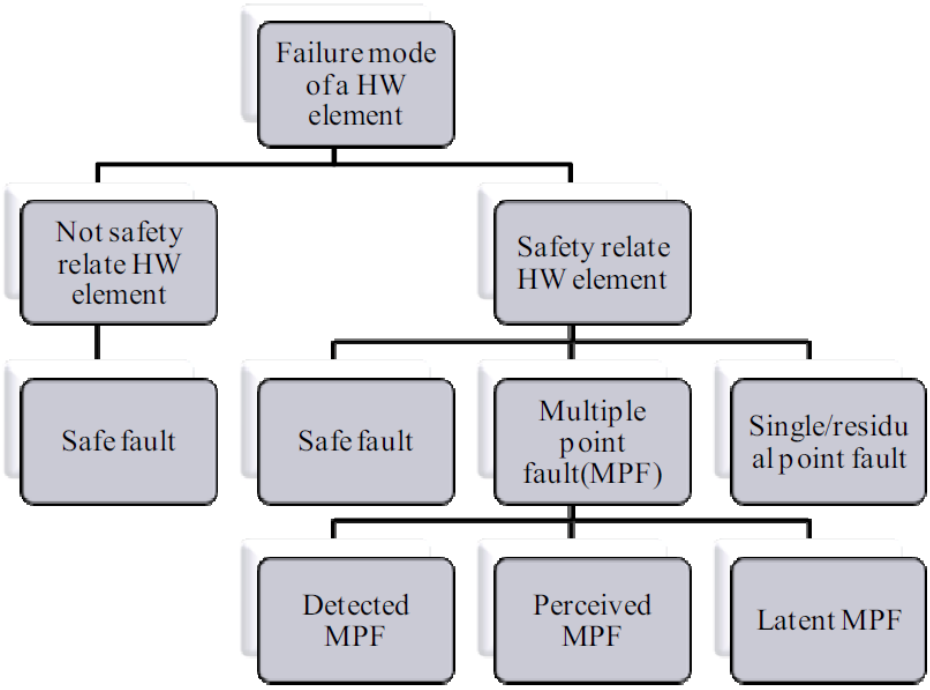

Figure 1.5. "Failure modes classification of a hardware element" [4]

Jung *et al.* in [4] report the following definitions from the standard:

- "**Safe fault**: fault whose occurrence will not significantly increase the probability of violation of a safety goal" [4]
- "**Multiple point fault**: one fault or several independent faults that in combination, leads to a multiple point failure (either perceived, detected or latent)
    - **Perceived**: This fault is deduced by the driver without detection by a safety mechanism within a prescribed time.
    - **Detected**: This fault is detected by a safety mechanism to prevent the fault from being latent within a prescribed time.
    - **Latent**: This fault is neither detected by a safety mechanism nor perceived by the driver." [4]
- "**Single point fault**: fault in an element which is not covered by a safety mechanism and where the fault leads directly to the violation of a safety goal" [4]
- "**Residual fault**: portion of a fault which by itself leads to the violation of a safety goal, occurring in a hardware element, where that portion of the fault is not covered by existing safety mechanisms" [4]

For each of these definitions it is possible to evaluate the corresponding failure rate. Given these characterizations, the procedure described by the standard concerns the evaluation of two specific metrics: the **Single Point Faults Metric** (SPFM) and the **Latent Faults Metric** (LFM). These systems of measurement are representing, respectively, the robustness of the element under consideration to single point faults and to latent faults. This robustness is guaranteed either by safety mechanisms of by design.

There is another important metric to take into account, called the **Probability of Dangerous Failure per Hour** (PFH). The **Failures in Time** metric (FIT) metric is obtained in this way: 1 FIT corresponds to a value of 10^{-9} of PHF.

Kafka in [5] describes also the way to evaluate the mentioned metrics. What is important for this thesis work, is their relationship with the ASIL levels, and this connexion is clear from the figure 1.6:

ASIL	SPFM	LFM	PFH
A	--	--	$< 10^{-6}$
B	$\geq 90\%$	$\geq 60\%$	$< 10^{-7}$
C	$\geq 97\%$	$\geq 80\%$	$< 10^{-7}$
D	$\geq 99\%$	$\geq 90\%$	$< 10^{-8}$

Figure 1.6. "Target values of SPFM, LFM, PFH dependant from ASIL A, B, C, D" [5]

1.4. Introduction to Deep Neural Networks

Based on Machine Learning (ML) and historical data, Artificial Intelligence (AI) consists in implementing and deploying algorithms for machines to solve and act on problems by themselves. Deep Learning is one of ML methods whose purpose is to extract features from data. Neural Networks were mentioned for the first time in the 1940s [6], with the purpose to emulate the human brain system to find the solution to general learning problems. In the 1980s and 1990s this methodology was sufficiently common. Nonetheless, it

experimented a decline basically due to the lack of computational power. Then, since 2006, Deep Learning has become widespread, thanks to the comparison in the market of some large-scale annotated data, to fast progress in designing of systems able to provide high computational power also exploiting parallelism (i.e. GPU). Moreover, some problems were more acceptably handled, like the management of a good initialization and the reduction of overfitting.

Furthermore, with the advent of DNNs, it was possible to obtain better results thanks to their deeper architectures: using an augmented and larger amount of layers they started to be able to capture higher levels of patterns and this can be translated in the ability to learn more complex features.

A Deep Neural Network is usually made by different layers. Commonly, it is composed by an input layer, followed by multiple intermediate layers (named hidden layers) and an output layer. Each layer transforms the data to create a new representation (in a hyperplane) and processes each complexity in turn. In each layer there is a certain number of nodes (artificial neurons), each node has an activation function, which is a non-linear function (i.e. max[0, *value*]) of a weighted sum of the inputs. Depending on the number of mutual connections between nodes, layers can belong to different types. The activation functions are the way in which learning happens: it is necessary that they are non-linear to guarantee the learning process. There exist a lot of different types of activation functions. One common weakness of Neural Networks in general is that they can’t be explained, so we can consider them fully a black box: this happens mostly because of the non-linearity which is present thanks to the activation functions.

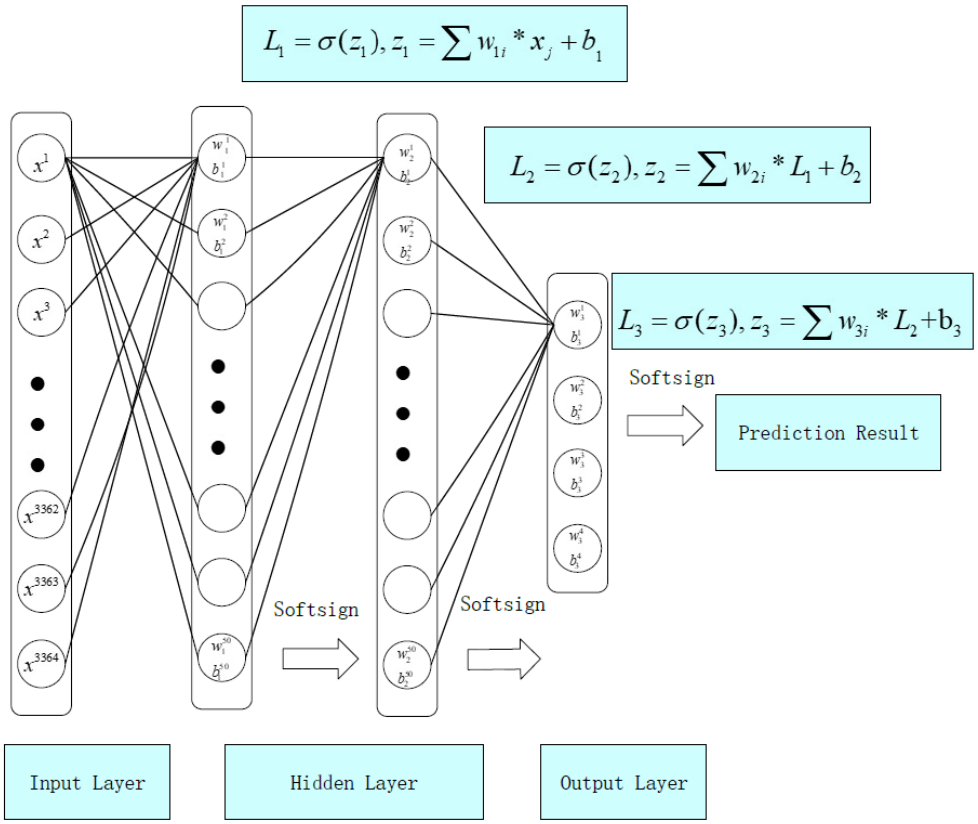

Figure 1.7. An example of a DNN Architecture from [7]

Going into details and referring to figure 1.7, it is possible to briefly describe the operation of a DNN. The learning knowledge is situated in weights and biases of neurons.

$$\sum_{j=1}^{L} g(\mathbf{W}_j \cdot \mathbf{X}_i \ + \mathbf{b}_j) = \mathbf{t}_i, i = \{1, 2, \cdots N\}$$

Where:

- $\mathbf{W_j}$ is the general weight vector;
- $\mathbf{X_i}$ is the input vector of features;
- $\mathbf{W_j} \cdot \mathbf{X_i}$ is the inner product between weights and inputs;
- $\mathbf{b_j}$ is the bias of each layer;
- **g(x)** is the activation function;
- $\mathbf{t_i}$ is the output vector;
- ***N*** is the number of layers.

To determine weights there are basically two phases: Training and Validation. Briefly, in the first phase, thanks to a training dataset, weights are evaluated. Then in the Validation phase, with a disjoint dataset the model is checked. Then if it is sufficiently convincing, the model is considered ready to be put into operation: this phase is called inference. DNNs, as just described, have a particularly parallel structure. To perform efficiently their typical kind of processing they are needed high computational capabilities, due mainly to the increased number of filter, neurons and parameters adopted, to provide a significant improvement in performances.

DNNs have found suitable for computer vision tasks. During the years, thanks to the lowered price of expensive processing hardware, to the growing processing capabilities and to the increasing amount of data that is started to appear online, it was possible to use DNNs with larger data sets and in real-life scenarios as Autonomous Driving.

It is possible to notice the hierarchical structure of layers in modern DNNs. One branch of DNNs that are actually deployed into self-driving cars, are Convolutional Neural Networks (CNN): in this particular type of network the first computation happens in convolutional layers (that are a certain number, from 3 to few tens), performing multi-dimensional convolution calculations. These layers are in charge to apply a filter to obtain the main features present at the input and to describe a feature map. Usually, there is a certain number of fully connected layers, that are dedicated to classification purposes. These kind of layers can be of Pooling type, extracting the local maximum value to be forwarded, or of Normalization type, averaging the inputs.

Concerning Computer Vision, the problem to which this work is oriented is Object Detection and Localization. So, the aim is, starting from data collection coming from sensors (mostly cameras), to assess where and what kind of objects are present in a certain image. Current algorithms are able to provide bounding boxes around objects that are detected and to deliver also the so called *Confidence Score*, assessing the probability with which the detected object

belongs to the provided category. Then, based on the results of Object Detection and Localization, path decisions are taken.

1.5. Deep Neural Networks in Safety-Critical Systems

A critical requirement for Safety-Critical systems based on DNNs' learning is the estimation of how much trust should be put in DNNs' output. Definitely, in applications like Autonomous Driving, even a single misprediction can have catastrophic consequences. Henne *et al.* in [8], present a Benchmark Analysis of Estimation Methods. They assert that it is very important to minimize the amount of mistakes made by such systems, and one way to do so is to improve performances, in terms of accuracy, as much as possible. The other side of the medal is that it is needed a certain degree of awareness concerning the reliability of the provided outputs: mistakes and errors can lead the system to different kinds of scenarios, with different criticality in terms of consequences. Regrettably, the standard method to evaluate the confidence when dealing with DNNs is based on the outcome of a typical final layer, called SoftMax, which is providing the class to which the detected object belongs and a so-called confidence score. This score is usually read as a posterior probability, providing a relative mark of the considered output with respect to all the other outputs that are being produced. So, the confidence score cannot be considered as reliability score, and this is one of the big challenges of this field. The problem is that the SoftMax score tends to give an overconfidence. This is particularly true in two cases:

- **Out-of-Distribution (OOD)**: When the input is not very similar to the training data. Out-of-Distribution Detection is a kind of analysis which aims to minimize the misclassifications in outputs given by a DNN. This analysis' purpose is to try to find kinds of data that in some circumstances can be at the input of the network and are hardly different with respect to the ones on which the network was trained on.
- **Adversarial example**: When the input is crafted to attack the network.

Chapter 2
General Reliability Evaluation Framework

In the second chapter of this thesis work there will be a brief analysis of the impact of general errors on the classification accuracy of the inference phase of Deep Neural Networks, followed by a case study highlighting the fact that each system should be completely characterized to be safely put into operation. Then there will be a description of the most common reliability tests, and finally evaluation of criticality of errors happening in Deep Neural Networks execution.

2.1. Impact of General Errors on DNNs' Accuracy

Ozen and Orailoglu in [9] present an experimental analysis regarding the impact of general errors, assumed being bit flips during the computation of the activation functions or mistakes affecting weights and bias. The purpose is simply to show the importance of the topic because of the fact that human lives are involved.

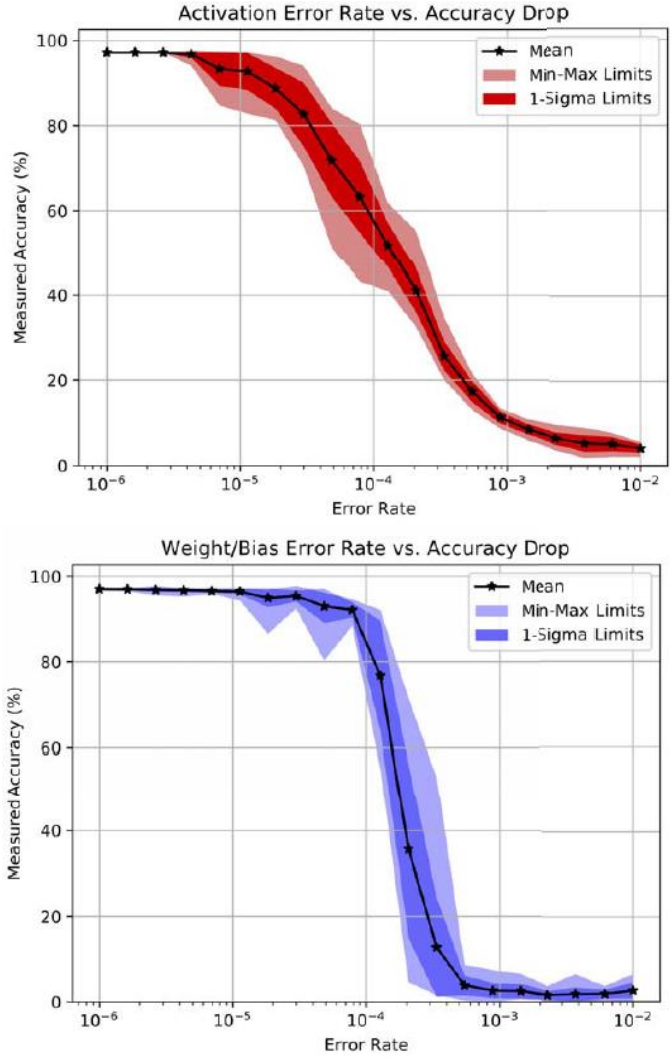

Figure 2.1. "The error impact on the DNN accuracy" [9]

From the results shown in figure 2.1, it is possible to infer the existence of a threshold for the error rate: when it is exceeded, it occurs a high degradation of the accuracy and therefore it is very likely to violate safety critical constraints.

Moreover, errors affecting weights and bias are more dangerous because of the practise of reuse of these values, consequently leading to an increased error probability.

A popular structure adopted to perform the typical computation of neural networks is a Systolic Architecture, made by a set of interconnected processing elements. In the case of DNNs, each processing element is only in charge to perform a multiply and accumulation (MAC) operation. A detailed description of the mentioned architecture will be outlined in Chapter 3. Concerning the impact of stuck-at faults, in [10], it is evaluated their impact on classification accuracy. In the analysis Zhang *et al.* focused only data-path faults since it was representing a significant fraction of the whole chip. Errors in memories are not taken into account because of the fact that they can be corrected, most of the times, thanks to the adoption of suitable Error Correcting Codes.

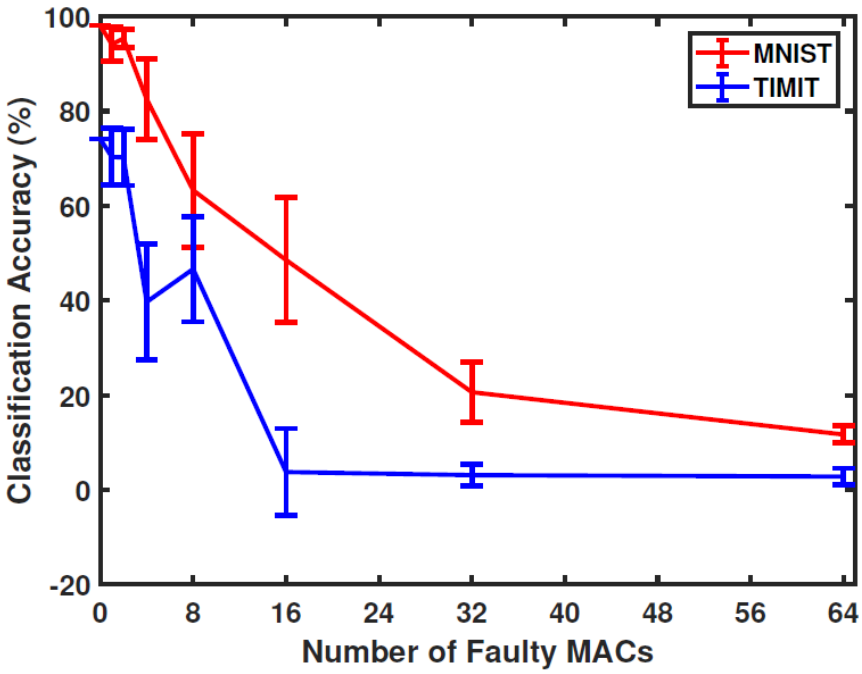

Figure 2.2. "Classification Accuracy Drop Due to Stuck-at-Fault MACs." [10]

In figure 2.2 are presented the results of the analysis of the impact of simulated stuck-at faults on classification accuracy: it is worth noting that even with little MACs affected there is a considerable drop in the accuracy.

2.2. A Case Study on Bit Transitions

Hanif *et al.* in [11] present a case study highlighting the fact that each system should be completely characterized to be safely put into operation. It regards one singular aspect: they distinguish the faults only in two kinds: transition of a bit value from 1 to 0 and the opposite case. In this investigation it is set a simulation in which there are memory faults, affecting network's parameters. The structure considered is shown in figure 2.3.

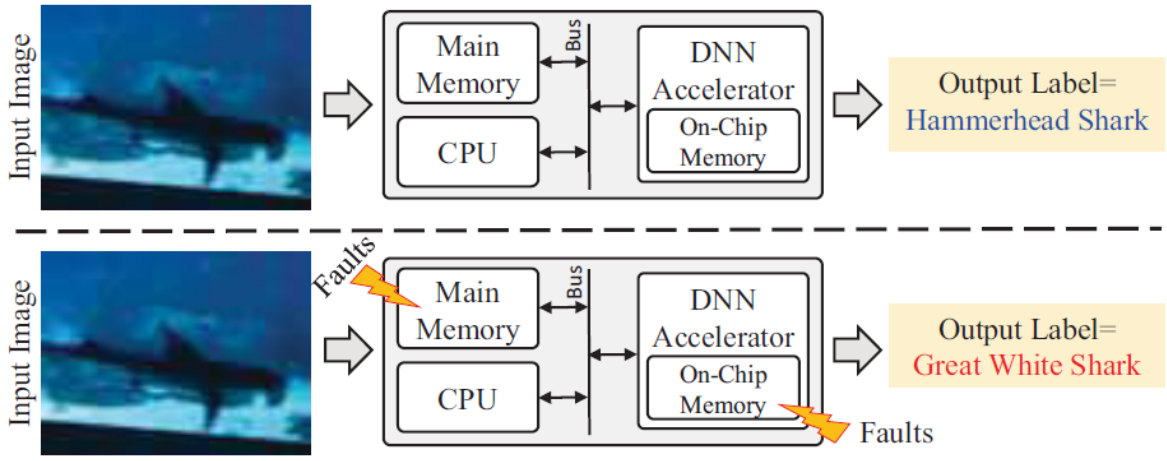

Figure 2.3. "Experimental setup for illustrating the impact of memory faults in DNN execution." [11]

To represent both weights and inputs, it is assumed also to use a 32-bit floating point precision, whose structure is shown in the figure 2.4.

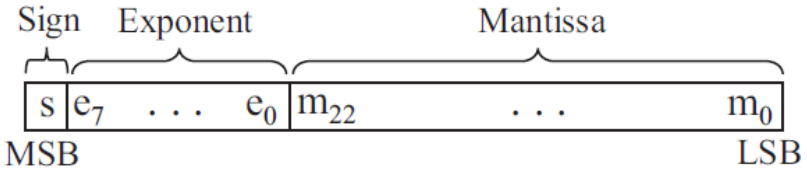

Figure 2.4. "Single precision floating point storage format used in our DNN design**."** [11]

In this study, to mark the difference of the impact of the bit flip when it happens in different locations in the word, the injection of faults is done separately and analyzing the impact of the failure of one bit at a time. The outcomes of this analysis are exhibited in figure 2.5.

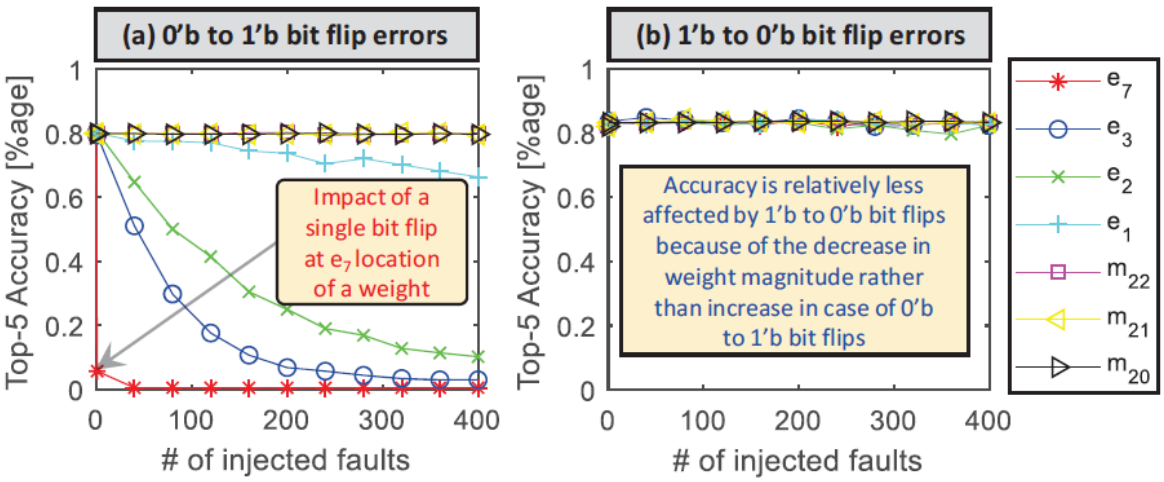

Figure 2.5. Impact of bit flip errors, distinguishing different transitions [11]

They performed a thousand of tests to compute this accuracy. The first outcome is that the impact of any kind of fault drops as the bit significance goes down. The second one is obtained comparing the impact of transitions: the bit flips from 0 to 1 have a drastic effect on the accuracy of the network if related to bit flips from 1 to 0. The reason in this particular case of the denoted behaviour is that when it happens a 0 to 1 transition in an exponent bit, the result is a large growth of the output, and this mismatch results in a misclassification.

The aforementioned case study reveals one kind of analysis that is possible to do to better characterize the consequences of different kinds of faults in a specific application.

2.3. Reliability Tests

Reliability tests are mainly implemented through the so-called fault-injection campaigns. In general, these operations require a huge number of simulations and therefore they are often expensive in terms of time and energy consumption.

In literature there exist two main categories of fault injection techniques:

- **Hardware-based techniques:** in this case faults are directly injected in the target hardware; hence these methods are more realistic and are able to provide true-to-life results.

- **Software-based techniques:** here hardware faults are simulated at an abstract level; they are able to provide sufficient accuracy and in general they are cheaper with respect to the previous ones.

2.3.1. Neutron Beam Experiments

The first methodology to test the reliability, mentioned by Dos Santos *et al.* in [12], is related to neutron beam experiments. The purpose of this experiments is to characterize the behaviour of the chip with respect to soft errors.

The whole device under test is irradiated with a controlled neutron beam, hence in a realistic way every kind of fault can arise. Conceiving this class of experiments, it must be taken into account the conventional assumption that is always made when dealing with high reliability in the field: Faults occur one at a time (instant by instant). Typically, the neutron flux used in this kind of experiments is 5-10 orders of magnitude higher with respect to the actual (terrestrial) one at the sea level. This detail allows the characterization of periods of time that are much longer with respect to the periods in which the chip under test will be employed in reality.

For this kind of reliability characterization, it is needed a golden (fault-free) evaluation of each output and a comparison with the output of the device under test. Whenever a discrepancy is found, the situation is underlined as altered by a Silent Data Corruption (SDC).

There exists another kind of error that can happen and it is called Detected Uncoverable Error (DUE). As described by their name, these errors can be discovered by hardware or software means, but it is impossible to correct them. An example of DUE is a crash of the program. During neutron beam experiments it is possible to insert watchdogs with the aim to find DUEs. At the end of the experiment there is a post-processing phase in which outcomes are translated in a standard way, measuring the Failures In Time (FIT) rate, as stated by the standard ISO 26262.

2.3.2. Fault-Injection Framework

One limitation of neutron beam methodology is that the effective detection of a fault happens only in the case it is shown at output. So the fault propagation process is not clearly observed. To analyze this other particular case, there exists another kind of experiment called fault-injection, mentioned by Dos Santos *et al.* in [12]. It is software-based and there exist many open source programs performing properly this task.

The idea is to put in a casually chosen register a random value, in a randomly picked moment. The choice of where to inject the fault can be done based on an uniform distribution, considering the number of bits used to represent weights or values, as proposed by Neggaz *et al.* in [13]. To better emulate hardware faults propagation, random values are injected at low level: transient faults may result in a bit flip that, if propagated to memory, can cause the storage of a wrong value.

Bosio *et al.* in [14] use a Fault Injection Framework to study the impact of permanent faults, that can easily be generalized to each kind of fault whose effect is to determine a bit flip. They consider a context in which the fault location is defined punctually by a field called *Flo* in which they are specified in turn the correspondent faulty layer, the edge connecting the faulty node to that layer, the faulty bit in the weight representation and the polarity of the stuck-at. A pseudocode representing the injection is presented in figure 2.6.

```
1 run_CNN (CNN, golden_prediction);
2 for (i=0, i < FLo.size(), i++) {
3   inject_fault (Flo[i], CNN);
4   run_CNN (CNN, faulty_prediction);
5   compare (faulty_prediction , golden_prediction);
6   release_fault (Flo[i], CNN);
7 }
```

Figure 2.6. "Fault Injection Pseudo-Code" [14]

At the beginning there is a golden evaluation of the faulty-free network and, after that, the fault injection process is induced. Actually a list of locations where to inject the fault is provided, then for each component of this list an

execution of the faulty resulting network is run and the result is compared with the faulty-free one. After this step, it is possible to make a classification of errors, and it is possible to identify a masked phenomenon as a situation in which there is no difference between the golden prediction and the faulty one. In case of a difference, Bosio *et al.* in [14], decided to consider as a safe error an error in which the confidence score of the mainly detected element is within a variation of ± 5% with respect to the golden prediction. Otherwise, it is an unsafe error.

2.4. A Fault Vulnerability Evaluation Methodology

Lotfi *et al.* in [15] introduce an interesting **parameter-based** Fault Vulnerability Evaluation Methodology, based on the assumption of using a Neural Network that provides a confidence score for the detected bounding boxes using logistic regression. It can be easily extended to Neural Networks using other methods to determine the confidence score of the boxes. As it is possible to see in Figure 2.7, it is assumed to have the possibility of running in parallel, receiving the same inputs, a Golden Network – Fault free and a Faulty Network, in which it is simulated the arising of a fault.

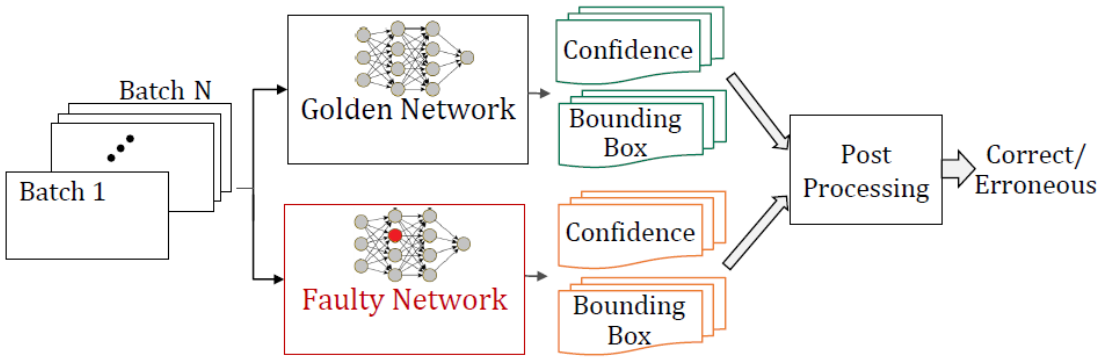

Figure 2.7. "Evaluating the impact of faults on the inference object detection network accelerated by GPUs" [15]

- In this process there is a bit of post processing done in order to adjust the output of both networks to better provide an evaluation. The first step aims

to get rid of the bounding boxes having a confidence lower than a threshold, to be properly set.

- The second step is dedicated to merge bounding boxes that are discovered to represent the same object in the same region: it is defined a metric called Intersection over Union (IoU): it is the ratio between the intersection area of two different boxes, over the union area. The nearer is the value of IoU to the unit value, the better is the overlap between the two boxes taken into consideration. Again, in this case there is the need to set another threshold to decide whether to merge or not the two bounding boxes under investigation. Later the confidence scores of the two boxes are summed, and if this value is higher than a third specific threshold (as usual, to be properly established), then that cluster of bounding boxes is picked as a detected object in the final set.
- The rest of bounding boxes not passing the thresholds-analysis are discarded and not presented at the output.

This is the moment at which happens the comparison between the bounding boxes of the object revealed by the two networks. It is considered the overlap ratio, evaluated reusing the aforementioned metric (IoU): a fourth threshold value is set and based on it, a decision is made. The decision process is shown in Figure 2.8.

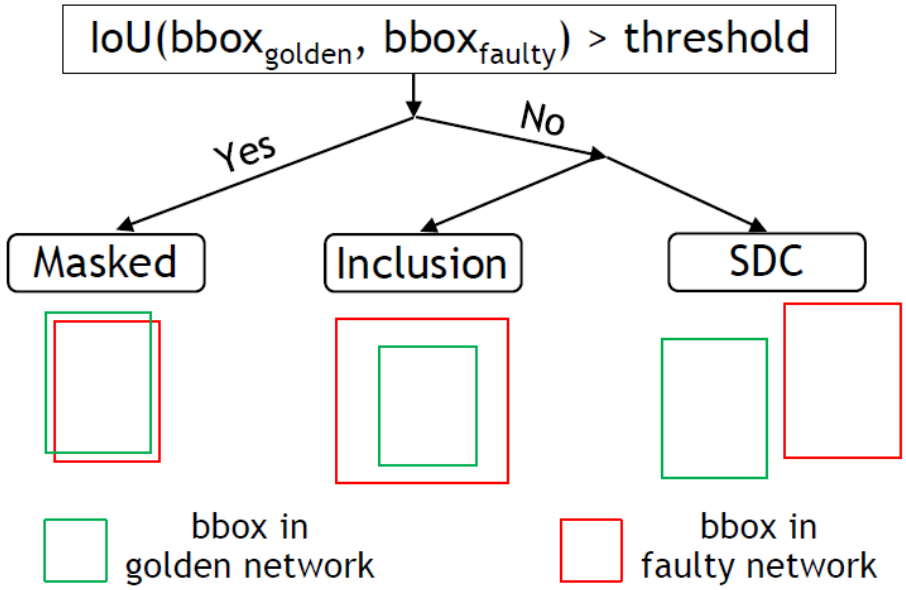

Figure 2.8. "Comparison outcomes for faulty and golden objection detection networks (bbox stands for bounding box)" [15]

The authors introduce basically three possible scenarios:

1. **Masked**: It happens when the IoU ratio is higher than the threshold. In this case the error has no effect on the network output, the object is correctly detected and classified and therefore it is a safe scenario.
2. **Silent Data Corruption (SDC)**: It happens in the case in which the IoU is lower than the threshold. Here the fault succeeded in generating an error that is translated into an incorrect output.
3. **Inclusion**: This is an intermediary situation, in which the IoU ratio is lower than the threshold but the faulty network at the end detects and classifies the correct object. This scenario can be classified either as safe or SDC: the trajectory planner in the autonomous vehicle can either avoid the obstacles or react in advance or with a certain delay.

Now, given the fact that this discussion is dealing with a strong safety critical application in which human lives are involved, the inclusion case should be included in the unsafe scenario. Some examples are shown in Figure 2.9.

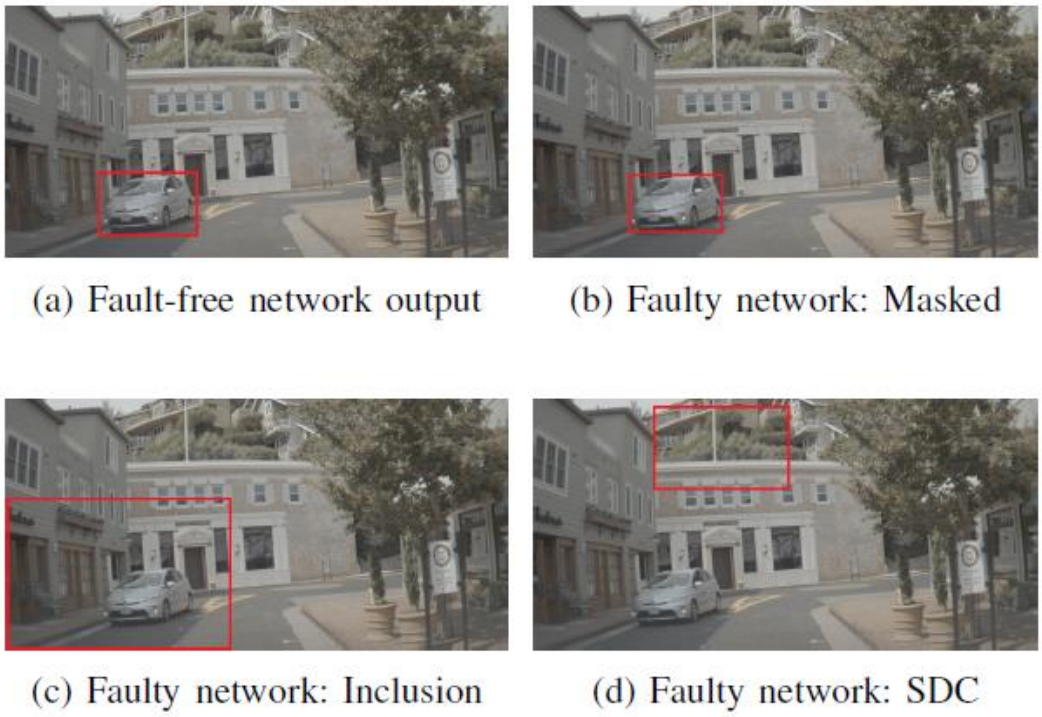

Figure 2.9. "Examples of fault effect on the output of object detection network" [15]

This methodology, anyway, is strongly dependent on parameters, that should be properly set.

2.5. Error Criticality Evaluation

Dos Santos *et al.* in [12] describe an interesting classification of errors, based on the evidence that errors are not all equal. Indeed, certain output errors can be considered as tolerable if the predicted object's classification and localization are adequately close to the fault-free output, that in this case is represented by the environment around the autonomous vehicle. The fundamental idea behind this classification is to combine metrics already known, characterizing the behaviour of Deep Neural Networks:

$$Precision = \frac{TP}{TP + FP}$$

Precision, or positive predictive value is the ratio between the number of True Positives (TP), that are objects correctly detected and classified, and the sum of TP and FP, where FP stands for False Positives, representing the number of objects incorrectly detected (assuming that a fault-free DNN is never giving a

FP as an output). Precision is the fraction of relevant instances among the retrieved instances and it indicates the amount of selected objects that are relevant. If the precision score is 100%, it means that all the detections made by the classifier are correct.

$$Recall = \frac{TP}{TP + FN}$$

Recall is defined as the ratio between the number of True Positives (TP) and the sum of TP and FN, where FN stands for False Negatives, representing the number of objects that are present but not detected. Recall is the fraction of the total amount of relevant instances that were actually retrieved and it indicates the quantity of relevant items that are selected. If the recall score is 100%, it means that all the objects in the environment were perfectly classified.

Based on a combination of the two metrics just mentioned, Dos Santos *et al.* in [12] separate errors in two classes: non critical errors and critical errors.

- The first class is the one labelled as **non critical error**, and it is characterized by the following conditions: Precision higher than 90% and Recall equal to 100%. The strict condition on Recall means that every object in the environment is actually detected, but the relaxed condition on the Precision value means that it is possible that an object could be incorrectly classified.
- The second class is categorized as **critical errors**, and it is denoted by the subsequent circumstances: Precision lower than 90% and Recall different from 100%. Thanks to the first circumstance there is an amount of non-existing objects identified and this can lead the autonomous vehicle to make superfluous stops. Moreover, the fact that the Recall is not perfect means that there exist real objects which are not seen by the vehicle and this can produce accidents.

Another important observation that can be done, as considered by Neggaz *et al.* in [13], is to distinguish the critical impact that a fault brings if it affects values or weights that are going to be wrongly stored in memory and the lighter effect of a fault affecting a combinational evaluation which is only made once.

Chapter 3

Permanent Faults Mitigation on Deep Neural Networks' Hardware Accelerators

In the third chapter of this thesis work there will be a description of some innovative techniques to mitigate the impact of permanent faults (stuck-at) generally in Hardware Accelerators. These methods regard the post-manufacturing phase. The structure of the chapter is the following: at the beginning there is a summary of a general methodology, then an error resilience analysis and finally some specific methods are detailed.

3.1. A Methodology for Designing Reliable AI Systems

Henne *et al.* in [8] show a perception chain dedicated for an autonomous driving system. Here it is possible to see implemented a countermeasure to include a possibly unreliable DNNs' based Computer Vision algorithm, into an autonomous vehicle. The core idea is to delineate borders of the unreliable part in a box called **safety-envelop**. An example is shown in figure 3.1.

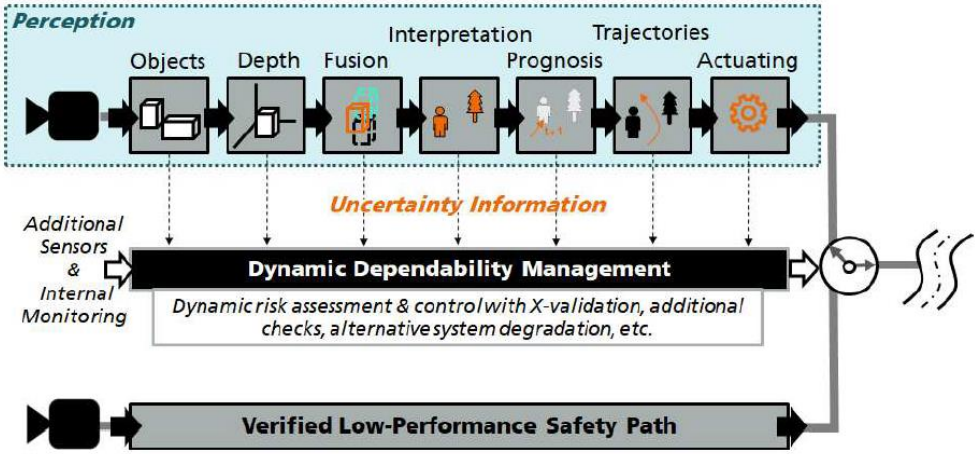

Figure 3.1. "Concept overview for utilizing uncertainty information of modular perception stages for dynamic dependability management" [8]

In each step of the perception, it is involved the use of a certain kind of DNN. There are basically two outcomes after each stage: a first one that is directed to the next step for further processing and a second one assessing its reliability, which is sent to a runtime monitor. Then online decisions regarding which information to trust are taken and in case of a decision to not trust that information, the Artificial Intelligence part is excluded and it is activated a Low-Performance safe path taking the overall system in a safe scenario.

Hanif *et al.* in [11] propose a general approach thanks to which it is possible to deal with design of resilient and reliable systems running machine learning and artificial intelligence applications. The procedures are divided in two big categories: Design-Time and Run-Time, and are described in Figure 3.2.

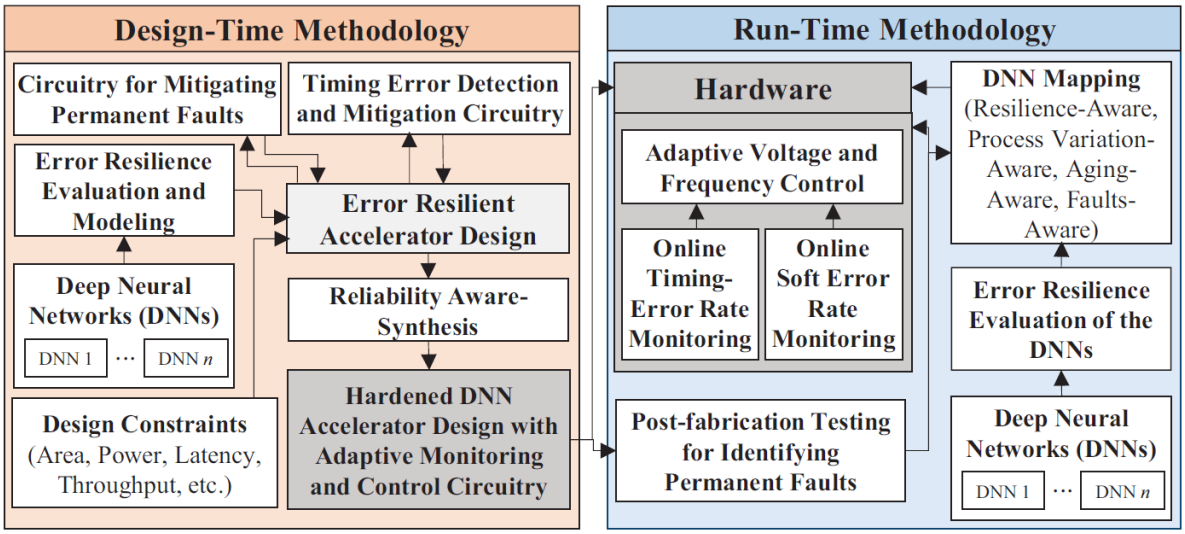

Figure 3.2. "Methodology for designing reliable systems for ML applications" [11]

The core problem in design time is to define an high reliable error-resilient Deep Neural Network Accelerator. It is strictly oriented to the kind of networks that will run on it. It is rendered an error resilience analysis focused on modelling the robustness of the network. Then from the outcomes of this analysis, together with design constraints a *zero*-accelerator is designed. Later they suggest adding additional circuitry to provide support for run-time reliability (i.e. timing error detection and mitigation circuitry). Finally the accelerator is developed, using a Reliability Aware-Synthesis technique. For what concerns reliability of memories, Error Correction Code (ECC)-based practices can be adopted and they can be really effective. The authors here do not recommend the use of redundancy because of the significant overhead introduced and since each element added is hungry of resources.

A detail which is included in this methodology is the fact that the outcome structure can sustain instruction-level redundancy or partial hardening of critical hardware modules.

Regarding the run-time methods here are suggested basically two approaches:

- **Fault-aware mapping of the networks**: It is needed a post-production permanent-faults identification, whose outcomes are intersected with the ones of the aforementioned error resilience analysis to apply a fault-aware mapping of the networks. This approach is exploiting the error

resilience analysis for offering as considerable performance and resource benefits as feasible.

- **Adaptive voltage and frequency scaling**: Approach based on a monitoring of soft/timing errors, trying to guarantee reliability trading off with performance and energy efficiency. This can be achieved applying adaptive voltage and frequency scaling. If required, it is possible to add Software redundancy.

3.2. Deep Neural Networks' Accelerators and Graphic Processing Units

It has already been discussed about the fact that DNN-based inference applications need a significant amount of computational power to be efficiently run. One possibility is to use a certain kind of accelerator. The first one that this analysis is taking into account is a Graphic Processing Unit (GPU). Indeed it is a family of architectures well-suited to improve the performances of such DNNs because those chips offer a good leverage of data and an efficient thread-parallelism, necessary for the kind of operations they have to perform, that as mentioned are multiplications and accumulations. GPUs are one of the enabling technologies to allow object detection and classification in the Autonomous Driving applications, since they can offer high-throughput computational performances. There is the need to assess the vulnerability to hardware faults to ensure that the safety requirements imposed by the standard ISO 26262 are fulfilled. The investigation is based on the evidence that any kind of fault arising in the GPU hardware may result in a failure in the network running on top of it, which on turn may cause an infringement of a safety requirement.

The basic internal structure of GPUs is shown in figure 3.3. It is possible to see that internally there is a partition into blocks called streaming multiprocessors (SM). Each of these cells is a computing unit that can run different threads in parallel, but each one owns dedicated memory cells. Then to schedule these threads it is needed only a simple control unit, since each thread can count on a

dedicated memory and so the sharing of complex resources is avoided. Moreover, to reduce latency, threads are not interacting with each other.

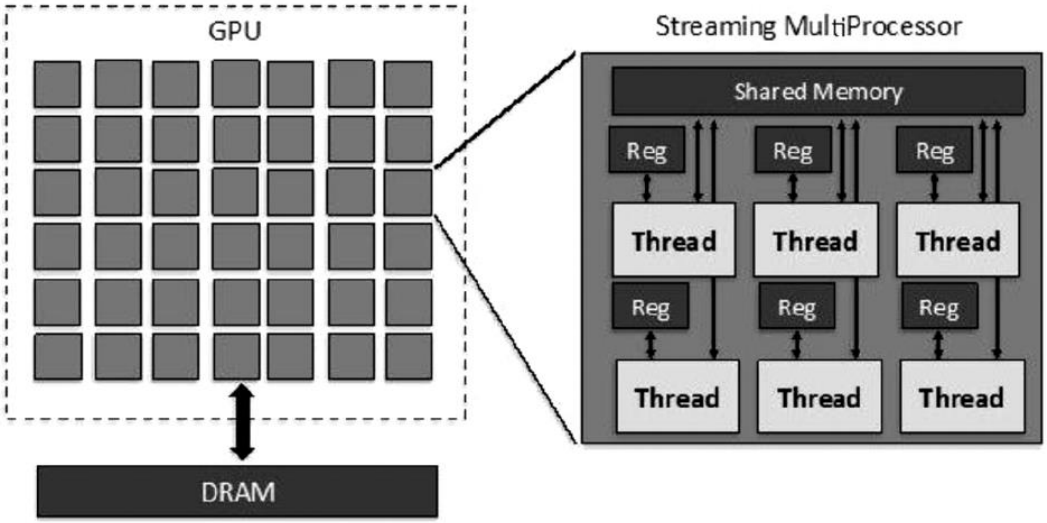

Figure 3.3. "Simplified internal structures of a GPU." [16]

Since GPUs were originally designed for non-safety-critical applications like the ones related to entertainment or video/image editing products, they may result particularly susceptible to transient errors [16]. Moreover, given their massively parallel structure and thus their need to sustain a considerable number of parallel processes, in the GPU there are large caches and register files, very vulnerable to transient faults.

The acceleration based on GPUs is only one way to reduce the computational load due to the presence of DNNs. Actually, there exists an increasing interest of the market in designing special purpose hardware accelerators in order to obtain better performances and simultaneously to be energy efficient.

There exists a list of properties that are common to all the DNNs computation, that can be exploited in the design of DNN Accelerators:

- The dependency between each computation is not strong: different operations can be executed in parallel;
- There is a strong dependency between used data: the reuse can be strategically adopted.

An accelerator involves basically a series of processing elements, like MACs, and a global buffer. Then it should be connected to a memory from which

transfer the data, like a DRAM, and a CPU usually performing scheduling and basically off-loading tasks from the accelerator.

A first point that should be taken into account is the difference between faults that can propagate in different ways: the ones affecting the datapath, that can only happen once, differently with respect to the ones in the buffer, that instead can be scanned multiple times because of the reuse.

Nowadays, almost all the training phase is made with floating point operations, and this is one reason because of the fact that the employment of GPUs has been so widespread. Then it is made a step called quantization, converting floating point numbers into relatively-small-precision integers (usually just made of 8 bits), because they are considered well-suited for an acceptable level of accuracy in inference [17]. This choice is improving energy consumption and area requirements.

Jouppi *et al.* in [17] propose an architecture of a DNN hardware accelerator called Tensor Processing Unit (TPU). It is employed mostly in Google Datacenters to accelerate the inference phase of some algorithms of Neural Networks. Since this chip is designed to be only an accelerator, it is more like a coprocessor, it needs the cooperation of a complete processor. For the purposes of this work, it is provided a brief description of the structure of this architecture.

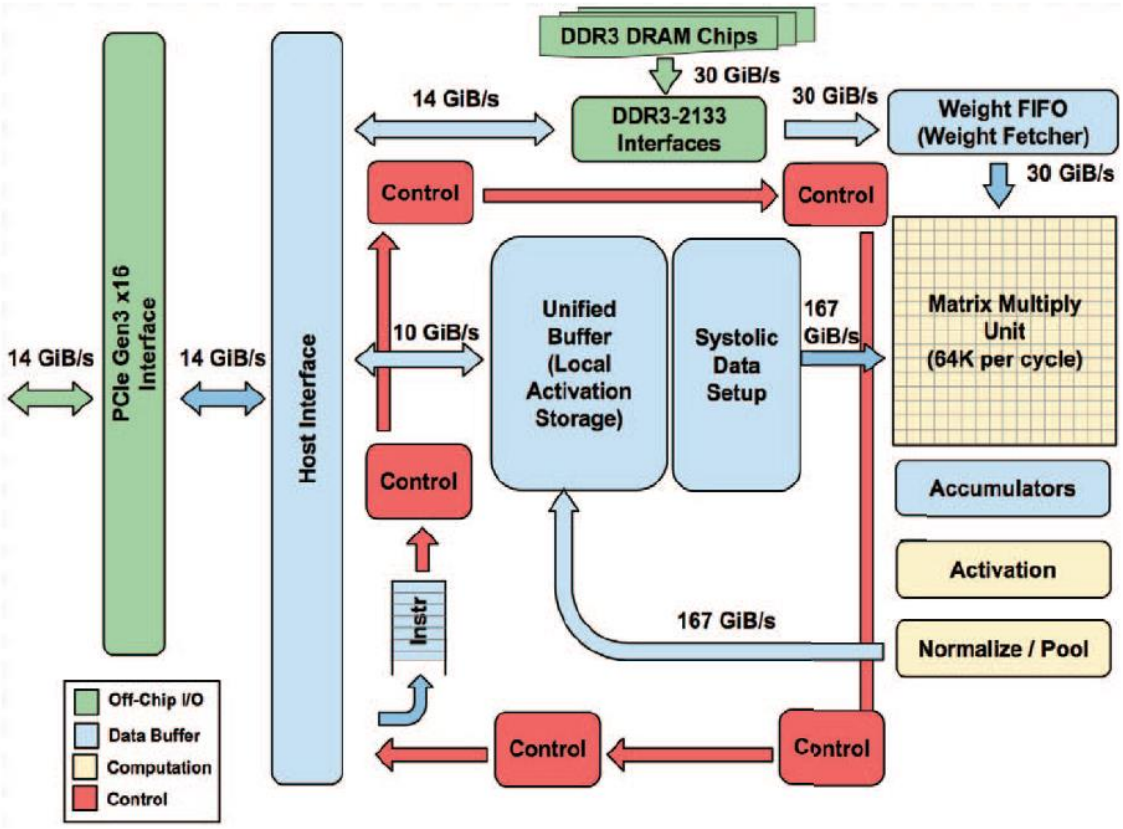

Figure 3.4. TPU Block Diagram [17]

In Figure 3.4 it is presented the block diagram. For the purposes of this work, it is worth noting that most of the computation happens in the Matrix Multiply Unit (yellow). The basic idea behind the architecture under consideration is to maintain the mentioned unit busy as much as possible, trying to overlap the Matrix Multiplication operations execution with the one of other instructions. Inspecting more in detail the considered fundamental unit, it is possible to notice the presence of a Systolic execution (Figure 3.5).

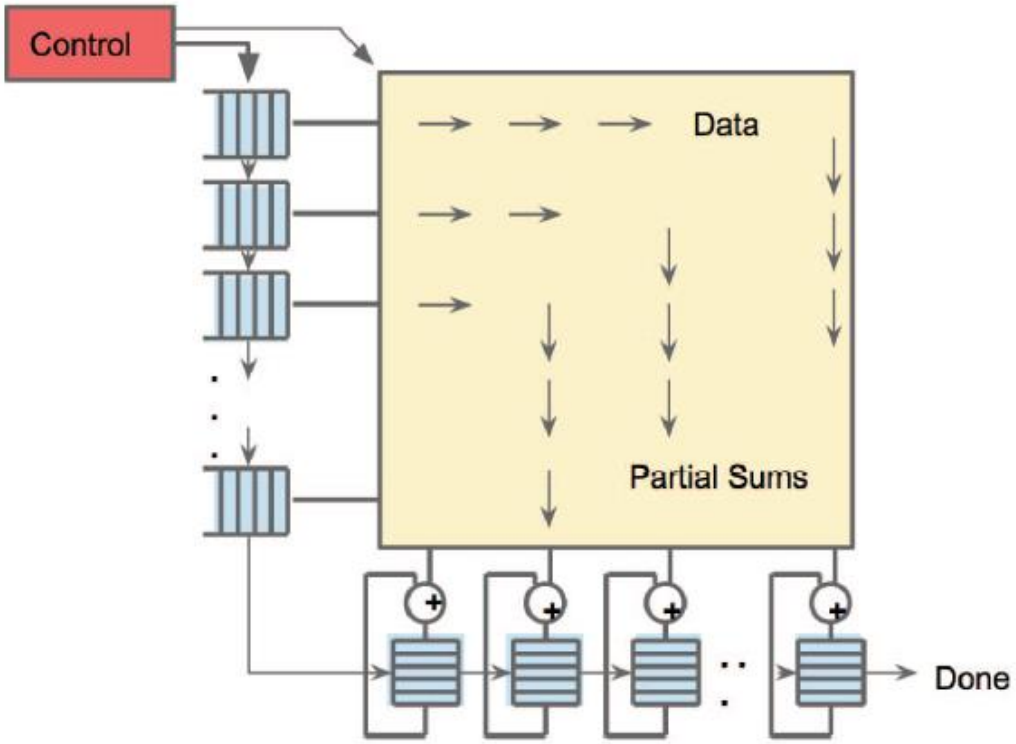

Figure 3.5. "Systolic data flow of the Matrix Multiply Unit" [17]

Data flow from the left, while weights are preloaded from the top: the multiply-accumulate process transfers through the matrix as a diagonal wavefront.

This architecture efficiently implements matrix multiplications and convolutions exploiting Systolic Arrays. Systolic Architecture studies can date back to the '80s [18], but are very efficient nowadays. More in details, a systolic architecture is made by a set of interconnected processing elements, but each of them is only able to communicate with its neighbours. In the case of DNNs, each processing element is only in charge to perform a multiply and accumulation (MAC) operation, before passing ahead the computed value. Therefore, data are streamed through the grid in a synchronized mode: in this way it is bypassed the need for computing and apply complex routing algorithms. Moreover, this architecture is able to provide energy efficiency, and this is due to the fact that reading's cost of inputs from memory can be amortized over several cycles.

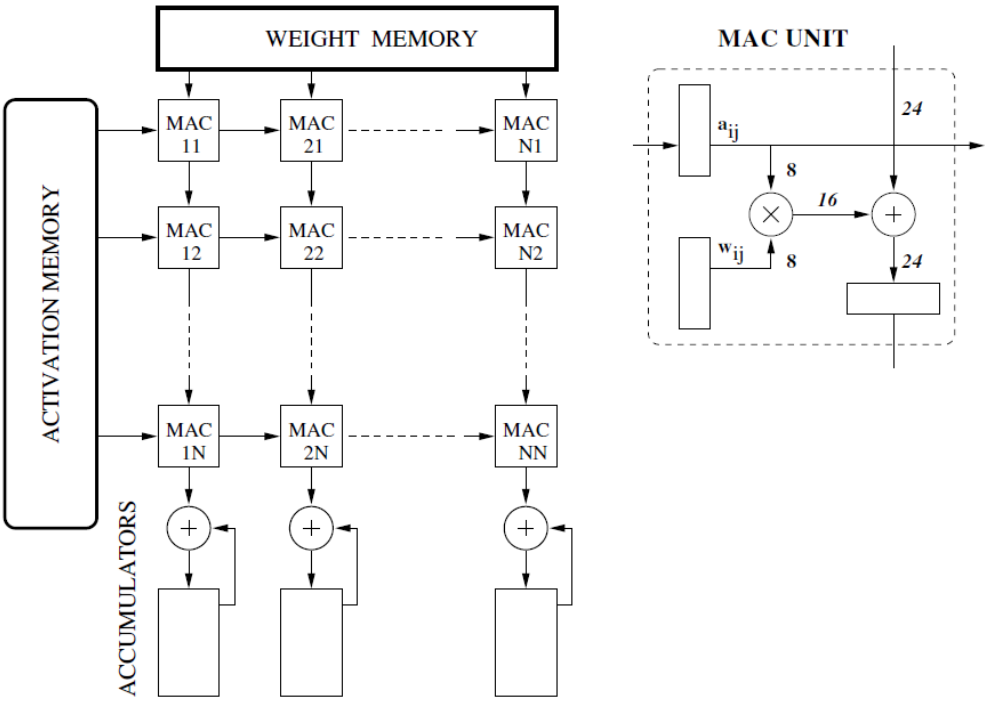

Figure 3.6. More detailed Matrix Multiplication Unit [10]

In figure 3.6 it is shown the systolic array containing *N x N* MAC units, able to perform matrix multiplications and convolution operations [10].

One advantage of this structure is that it is able to maximize the percentage of computing time to I/O operations' time. Different layers are processed one at a time. Different kind of analysis that are detailed in [17], show that the bottlenecks that can happen can actually relate more to lack of memory bandwidth with respect to peak of computing. It is worth saying also that this chip was designed considering also some requirements related to response time, since it was employed in Natural Language Processing applications, that are actually end-user-facing services. The important fact is that it is reduced the dependency with the host CPU: the TPU is able to entirely sustain the whole execution of Neural Network models; this feature can be crucial in the sense that dividing tasks between different chips it can be easier to build reliable architecture, as a set of safe components.

3.3. Error Resilience Analysis

Hanif *et al.* in [19] underline the essential necessity for an Error Resilience Analysis whose aim is to contextualize strength and weaknesses of a specific DNN. Therefore, based on this result, is it possible to apply the methodology described at the beginning of this chapter, but also to apply techniques, like pruning, whose purpose is to achieve an optimal gain in energy consumption or in execution time. Indeed, thanks to this kind of analysis it is possible to recognize probable candidates, ready to be modified (through suppression or optimization), without invalidating the global accuracy. The goal of the considered analysis is to examine the network resilience against errors of any sort. Thus, the authors divide the process in two steps: hardware level and software level error resilience analysis. For the purposes of the current work it is worth underlying the first approach.

One key assumption, often made dealing with general errors, is to consider the errors as independent and identically distributed. Therefore, the addition of errors caused by multiple and independent faults, for a sufficient amount of errors, can be considered as a Gaussian distribution. Consequently, the main concept is to test the error resilience of a specific network by electrically introducing a White Gaussian Noise (0-mean, as shown in figure 3.7) in particular locations of the network, modifying nodes' voltage.

Figure 3.7. "White Gaussian Noise (WGN)" [19]

Specifically, the procedure establishes to insert WGN at the output of each layer of the network and to evaluate one by one the modification of the accuracy. This technique is emulating random hardware errors but also the effect of a reduced-precision computing aimed to reduce the computational complexity. The results of an exemplary analysis are shown in figure 3.8.

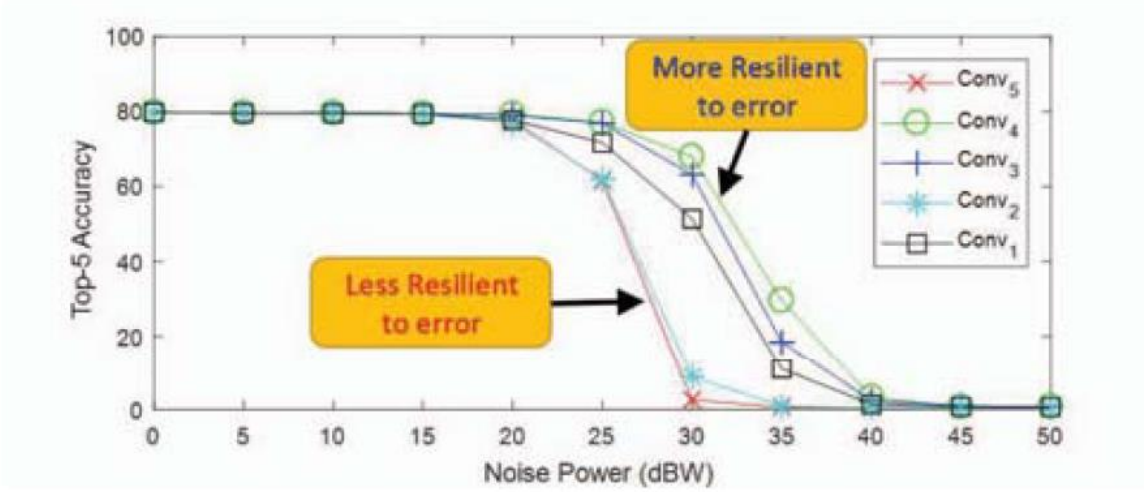

Figure 3.8. "Effects of introducing WGN at the output of different convolutional layers on the accuracy of the CNN" [19]

As case study, Hanif *et al.* used a VGG-f network architecture, whose structure is primarily made by eight layers (five convolutional and three fully-connected), as shown in figure 3.9. It can be trained for classifying images.

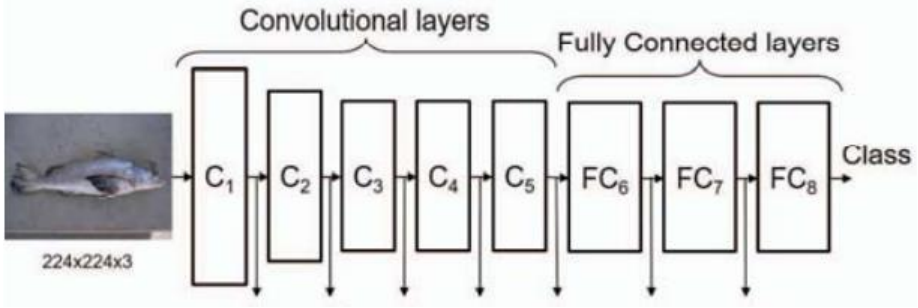

Figure 3.9. "VGG-f architecture. For illustrative purposes the activation, normalization, and pooling layers are not shown." [19]

The authors applied the analysis only to convolutional layers, since they are the most compute-intensive layers [19]. The important outcome is that, from this analysis, it is possible to identify the layers that shows more weaknesses. Based on these results it is now feasible to implement some countermeasures.

Moreover, they analyze also the consequences of the application of a non-0-mean WGN, but the optimal behaviour of the network is found to be shown in the case of 0-mean, which is the most likely case in the real world. Results are shown in figure 3.10.

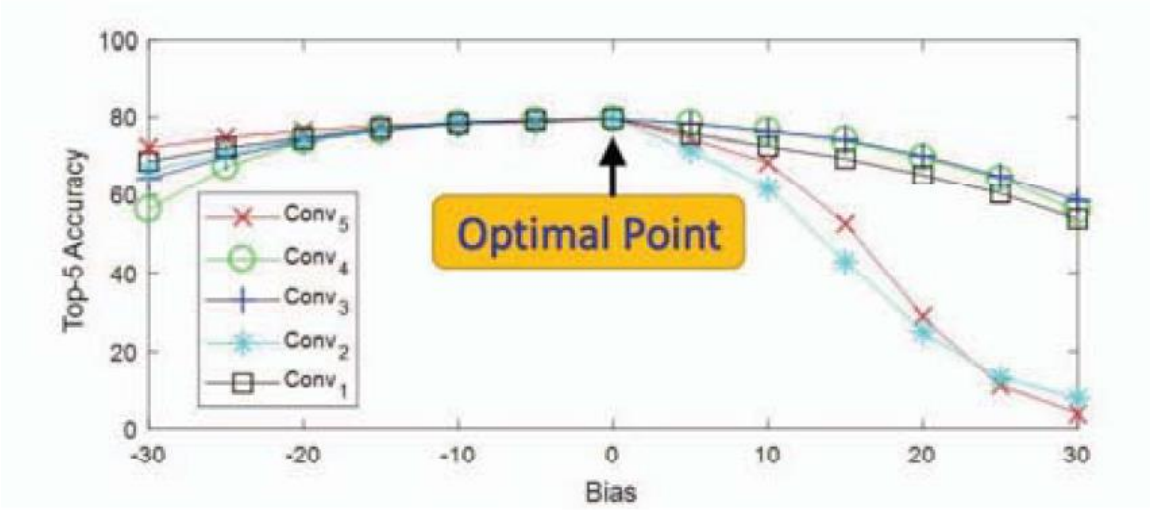

Figure 3.10. "Effects of introducing bias + 20dBW WGN on the accuracy of the CNN" [19]

3.4. Methodologies Reducing the Impact of Permanent Faults

Nowadays, permanent faults can be identified through specific tests after manufacturing. Anyway, the process of discarding all the chips showing one or more permanent fault can be a remarkable reason for yield losses. Zhang *et al.* in [10] propose two post-manufacturing methodologies for fault-tolerant design against permanent faults, which are fault-aware. These solutions are based on the concept of pruning connections, which was demonstrated to little affect the accuracy of DNNs. The methodologies under considerations are applied to the Google TPU architecture, previously described, simply adding minor modifications as bypass circuitry.

In the first works regarding systolic architectures from a fault-tolerant viewpoint the basic idea was to search for permanent fault and, once individuated, to simply deactivate the rows and columns of MAC units

containing faulty MAC. It was a very general solution conceived for each kind of application, nevertheless this solution was leading to important performances reduction.

Both the solutions proposed in [10] start from the observation that each weight is statically mapped exactly into one MAC unit: the idea is to take advantage of this kind of mapping to state the rules of pruning. The first solution is called **Fault-Aware Pruning (FAP)** and it is illustrated in figure 3.11:

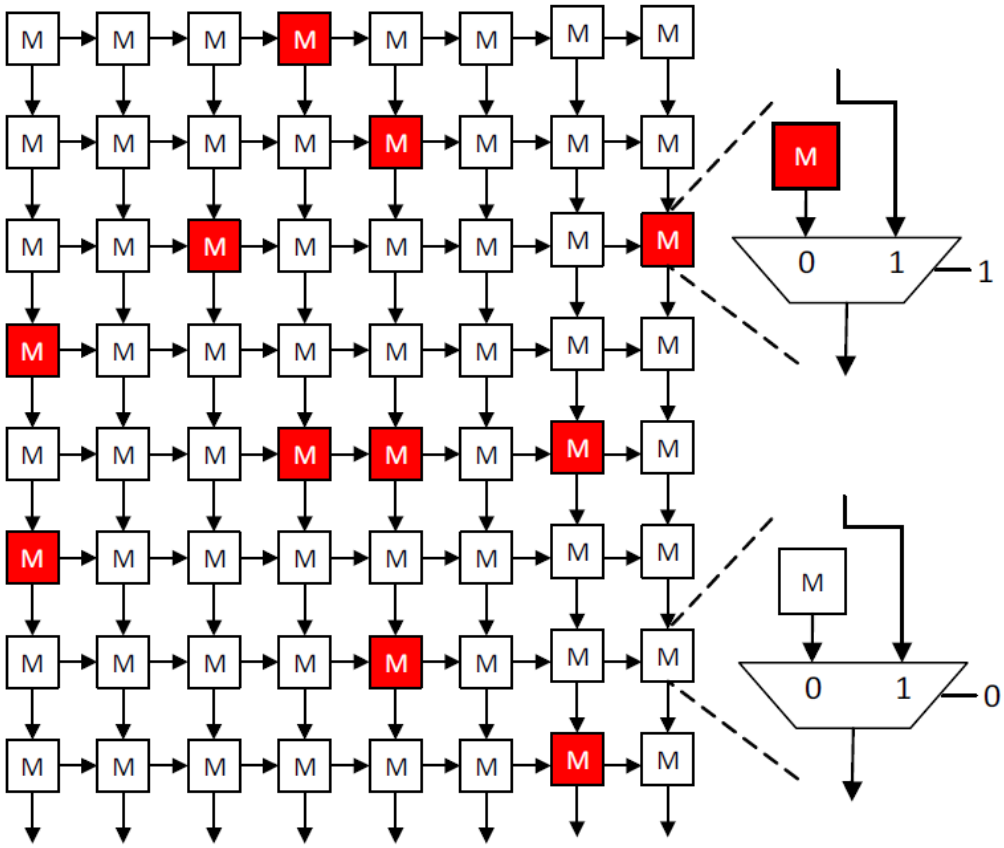

Figure 3.11. FAP Illustration [10]

Basically, given the information regarding (permanent) faulty MACs, coming from post-manufacturing tests, the main principle of FAP is to set to 0 (prune) each weight (or weights) which is mapped on a faulty MAC. The hardware bypassing circuitry is very simple, and it is exposed in figure 3.11. Obviously, this is representing an increasing in the area overhead, however it is estimated by Zhang *et al.* to be around the 9%, which can be acceptable.

The second solution is named **Fault-Aware Pruning + Retraining** (FAP+T) and it adds a second step with respect to the previous one: after the pruning

operation, there is the retraining of the not-pruned weights of the network, while the pruned ones are required to be zero.

Algorithm FAP+T()
Load the pre-trained DNN weights and TPU fault map;
Determine indices of pruned weights from TPU fault map;
Set all pruned weights to zero;
for *Training epochs* $\leq$ *MAX_EPOCHS* **do**
    Update weights using back-prop.;
    Set all pruned weights to zero;
end
return *Retrained model;*

Figure 3.12. FAP+T Algorithm [10]

This operation allows an improvement of the classification accuracy, but it requires a specific retraining part for each TPU. Some results from their simulations are shown in the figure 3.13:

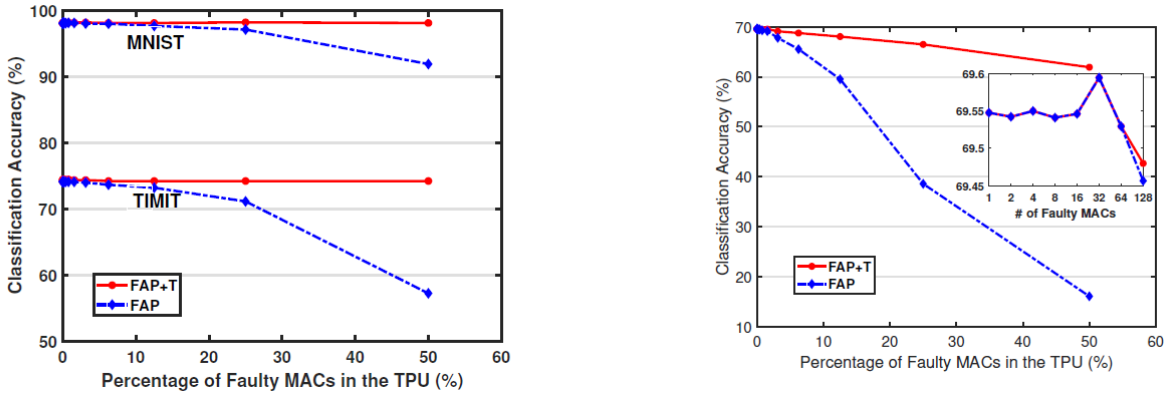

Figure 3.13. "Classification accuracy vs. Percentage of Faulty MACs using FAP and FAP+T for MNIST and TIMIT and AlexNet." [10]

A limitation of this work is provided by Chitty-Venkata and Somani in [20], which argue that it is not very effective as a methodology in case of column faults, as shown in figures 3.14 and 3.15.

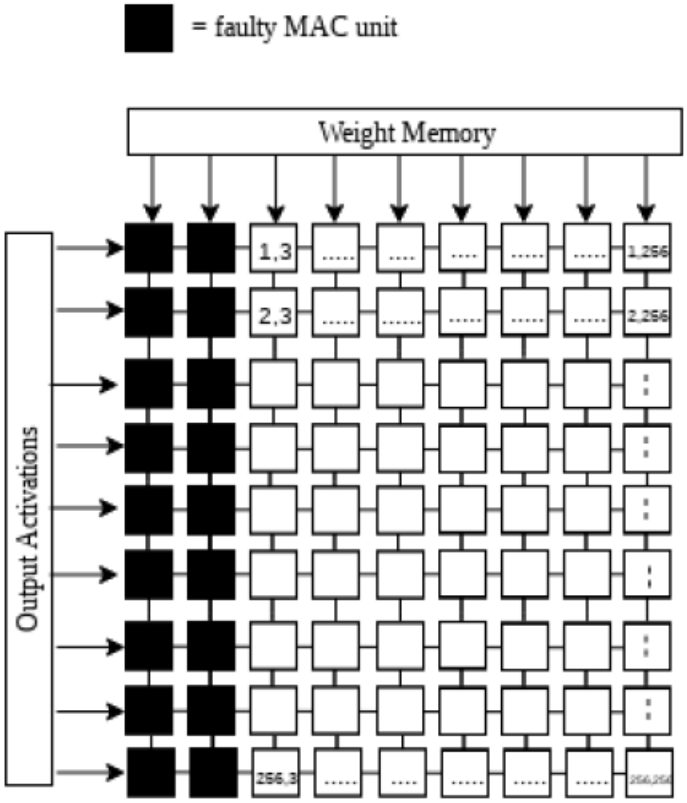

Figure 3.14. "Column Faults on Systolic Array" [20]

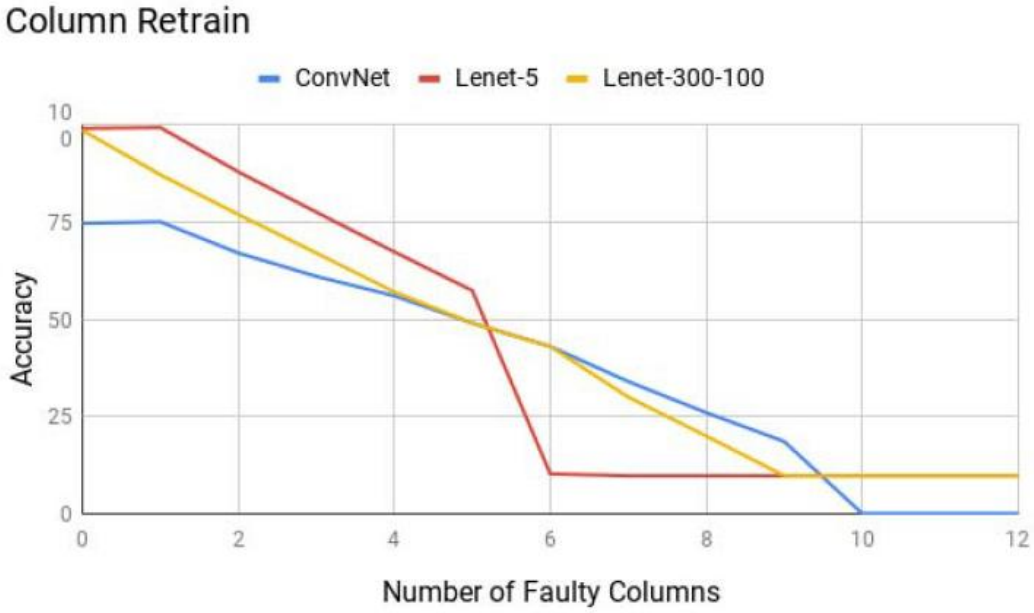

Figure 3.15. "Effect of Pruning Weights at Column Fault Locations plus Retraining" [20]

If a whole column is composed by faulty MACs, the result is a more dangerous situation with respect to the one that it is happening when there is an entire row made by faulty MACs. This is because pruning input neurons has little impact

with respect to the one made from an output node [20]. The idea exposed in this article is to apply some remapping techniques, as Matrix Transpose (for example in the case of low column faults to transform them in a low row faults) or Array Reduction (bypassing entire column through Matrix Transpose it was not possible to improve the accuracy, accepting a reduction in performance), avoiding to re-train the network.

Referring to the study, of Bosio et al. in [14] it is possible to state that it is very difficult to give general results regarding the impact of permanent faults on DNN-based architectures. Every combination of DNN and the corresponding hardware supporting its execution should be analyzed in order to find the most critical elements. Nevertheless, a valuable result is related to the underlined criticality of exponent's bits of the variables expressing the weights: since these bits are the most critical ones, one design solution could be to exploit redundancy applied only to the critical part of the system.

Chapter 4

Transient Faults Mitigation

In the fourth chapter of this thesis work there will be a description of the general impact of transient faults on DNN, then some innovative techniques to mitigate the impact of transient faults, firstly in Hardware Accelerators and then in Field Programmable Gate Arrays, followed by a brief observation about aging.

4.1. General Impact of Soft Errors on Deep Neural Networks

Neggaz *et al.* in [13] analyze the effect of transient faults on the accuracy of DNN models for safety-critical systems. The study is conducted injecting logical bit flips and the choice of where to inject the fault is done based on an uniform distribution. The network chosen for the current task is LeNet5, whose architecture is basically made of two sections: a feature extractor, built with two convolutional layers and a classifier, developed with three fully connected layers followed by a SoftMax one. The dataset used to train the network is *MNIST* (handwritten digits to be recognized, divided in 10 classes). Then it is studied the accuracy after fault injection. Since the aim is to provide a complete analysis, the evaluation of the effects of transient faults is divided in two categories:

- Errors affecting combinational circuits (called here Single Event Transient, SET): the impact is transient, indeed it is affecting once the specific current evaluation. To emulate this kind of errors in simulation, each transient event affects only the result of that given run.
- Errors affecting storage in memory (named Single Event Upset, SEU): here a transient event succeeds in flipping a bit in memory, then the result is to have a corrupted data stored. To simulate this behaviour, it is considered the impact of consecutive transient events, causing bit flips in turn, that should be seen by successive evaluations.

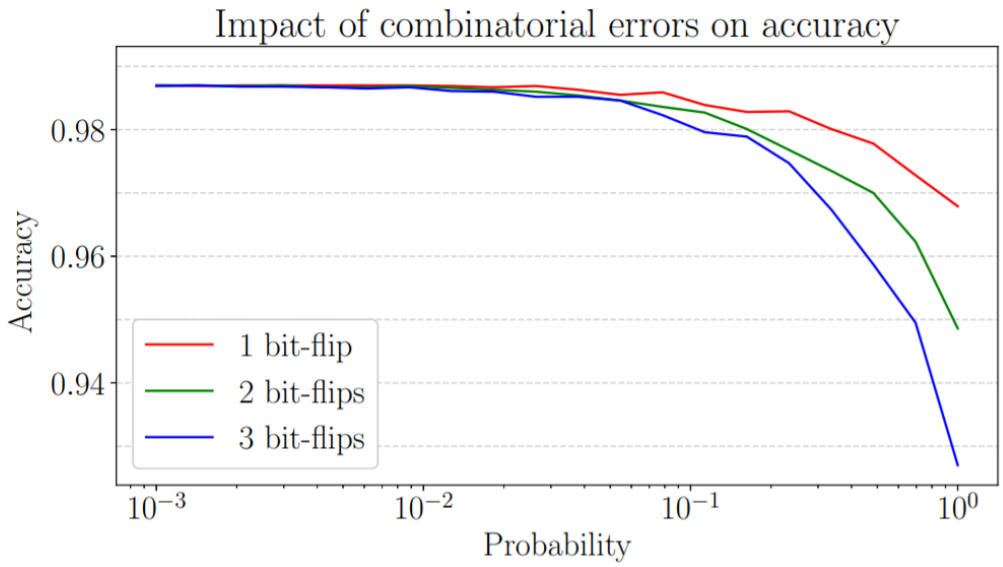

Figure 4.1. "Comparison between single and multiple event transients and their impact on overall network performance." [13]

In figure 4.1 it is shown the behaviour of the accuracy while rising the fault injection probability, in case of SET. The first observation that comes out is that the impact of injecting multiple errors on the drop in accuracy is more important with respect to the case of single error injection. Moreover, the accuracy starts to drop from a fault injection probability of 10^{-2}, but it is possible to notice that the global loss is still acceptable, and this is due to some natural DNN properties that make them strong with respect to SET [13].

Concerning the case of SEU, the cumulative factor of multiple errors in storage comes out and then its impact is more significant.

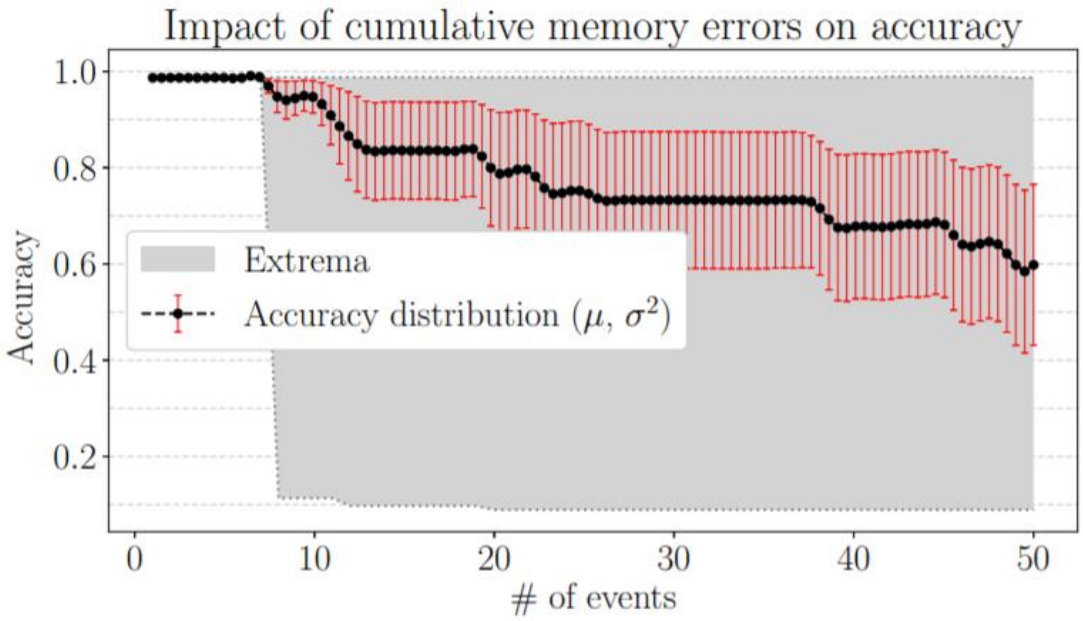

Figure 4.2. "Evolution of average accuracy (in black) when the number of errors augments. The dispersion and the extreme cases are shown in red and gray respectively." [13]

In figure 4.2 it is shown the impact of memory errors, which are translated in modifications of weights, on the accuracy of the network. It is worth noting that in the worst-case scenario the network becomes almost a random number producer, obviously violating safety requirements. Anyway, before 10 errors the network is still working properly.

Hence, this analysis is remarking that errors introduced in combinational circuits have a relatively small impact, and their effect is mitigated naturally from the network, while errors happening in memories have a bigger impact on network accuracy.

4.2. General Impact of Soft Errors in Hardware Accelerators

Li *et al.* in [21] highlight the different parameters that should be considered while searching for impact and consequences of soft errors in Hardware Accelerators:

- **Topology and Data Type:** the topology of the specific network under consideration should be carefully evaluated, since it is one major player in propagation of errors. Additionally, distinct data types can be differently affected by errors.
- **Bit Position and Value:** this parameter is strictly related to the data type, indeed it is possible to find more vulnerable bits depending on their location, and therefore it is possible to better protect them. One example here is the difference in relevance between an exponent's bit and a mantissa's one.
- **Layers:** errors happening in some layers can be more critical than bit flips occurring in other ones.
- **Data Reuse:** the last parameter is taking into account the fact that there are some computations that are made once and then reused, like weights. The criticality of an error can depend on the reuse factor of the word containing that fault.

4.3. Methodologies Reducing Impact of Transient Faults on DNNs' Accelerators

Li *et al.* in [21] continue proposing an interesting mitigation methodology for transient faults happening in DNNs accelerators. The analysis is started by excluding combinational logic's faults since they are usually much less critical with respect to memory components. Then given the nature of the accelerators, also it is omitted the contribute of errors in control logic units (that occupy only a very small part in accelerators). The method is called **Symptom-Based Error Detectors** (SED), and it is implemented via software. The mentioned technique monitors value ranges of the activation functions, and they are judged as symptoms to detect the possible faults: the idea justifying it is that if the output of an activation function acquires an high magnitude due to an error, it is very likely to obtain a Silent Data Corruption (SDC) at the output. The monitoring

task is assigned to a properly trained neural network, which is trained on a faulty-free operation to learn to judge the right ranges for the output of the activation functions. Then once trained, the detector can be implemented and executed asynchronously with respect to the nominal operation to check the ranges. As a result, the average precision obtained from experiments is 90,21% and the average recall is 92,5%.

Based on the Algorithm Based Fault Tolerance (ABFT) philosophy, firstly introduced in the 1980s by Huang and Abraham [22], Dos Santos *et al.* in [23] propose a strategy to reduce the impact of transient faults in GPU-based safety-critical systems. The core idea of ABFT strategies is to study the algorithm structure searching for a smart and effective way to reinforce it.

In this case the main concept is to encode the input data, to adjust the algorithm to work with these data and to introduce a check phase in the output to benefit from the encoding. In the case in which an error is detected, if there is much information, it can be corrected, otherwise the computation is stopped and re-launched. Basically, it is an extension of the ECC-based techniques to cases in which it is required computation and not just transfer of data.

The case in which faults happen at the beginning of the network, affecting the initial layers, could be one of the most dangerous because of the fact that errors can propagate in an uncontrolled way to the following layers.

Concerning the case under consideration in this paragraph, the idea is to strengthen the matrix multiplication operations, which are of crucial importance when running DNN-based algorithms. The methodology is shown in figure 4.3.

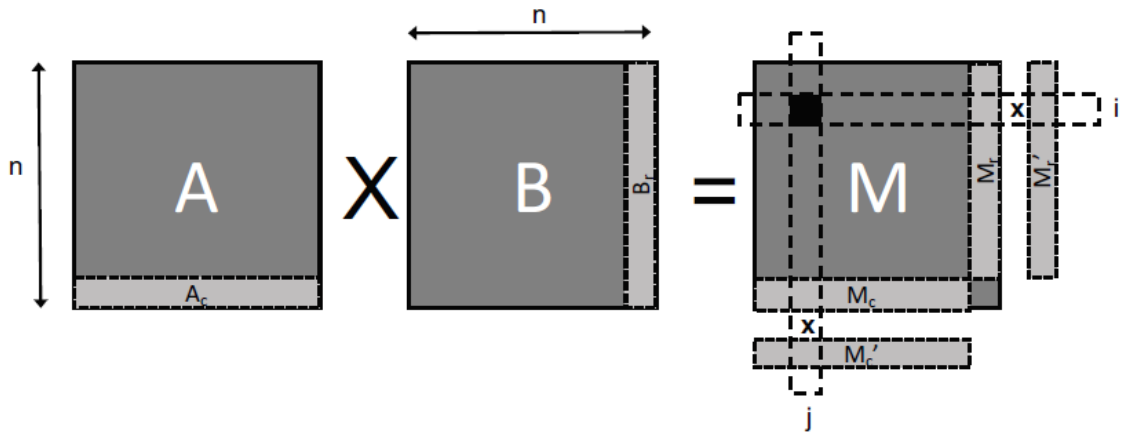

Figure 4.3. "Algorithm-Based Fault Tolerance for matrix multiplication" [23]

This algorithm comes from an adaptation suitable for DNNs from a more general algorithm presented by Rech *et al.* in [16]. Basically, input matrices are encoded before the actual operation, adding a row and a column checksum vector, evaluated as the ordered list of sums of the items present in the correspondent column or row. The original algorithm [16] planned to increase by one the number of row and columns, implementing the check row and column, to not lose accuracy. In this case the authors decided to replace one column and one row of the original matrix with the check vectors because when dealing with a specific hardware like GPUs, specific accelerators or FPGAs, they are specifically tuned on the dimension of the final application matrices. Thus, this choice is based on the desire to not increment too much the execution time. The price to pay is related to a decreased detection accuracy, that anyway it is proven to be sufficiently high to meet the standard requirements.

Then, once the computation is completed, the check vectors are re-evaluated in a separate calculation, and then it happens the comparison. In this particular case if it is detected only one error (mismatch) it is possible to correct it, otherwise a re-evaluation is required. This theory is aligned with the principal assumption that is at the fundamentals of the reliability theory that is that faults happen once at a time. The aforementioned result can be easily implemented to reduce the impact of soft errors in the field.

4.4. Selective Hardening on FPGA

An important category of chips which is promising to efficiently and safely implement DNNs based algorithms is the one represented by Field-Programmable Gate Arrays (FPGA). It is possible to underline, among others, some key characteristics of these circuits, like their affordability, low-power consumption and flexibility. The typical FPGA structure is suitable for the execution of parallel tasks. On the other hand, FPGAs are particularly vulnerable to radiation-induced errors, leading to critical consequences: more precisely if the transient error happens in the Static Random Access Memory (SRAM) in which it is stored the map of active routing connections, it is possible that the outcome is a modification of the configuration of some internal block (i.e. a Lookup Table (LUT) or a Block RAM (BRAM)).

Libano *et al.* in [24] propose an innovative reliability analysis of Neural Networks algorithms implemented through FPGAs. They start emphasizing some remarks similar to the ones found in every work evaluated by this thesis. The important preliminary statement declares that, given the innate nature of neural networks, not all the errors affecting the output are able to generate a meaningful impact on the execution. Thus, the error criticality is split in two categories for this study:

- **Tolerable error:** the output of the network is faulty but the behaviour is correct, meaning that the classification mechanism is concluded correctly;
- **Critical error:** output errors are severe enough to lead to misclassifications.

The research begins from an assessment on reliability of two different neural networks, whose training phase was performed in a faulty-free manner. The two networks under consideration are the *Iris Flower ANN*, composed by only two fully connected layers, and the *MNIST CNN*, made by 8 different layers (Convolutional, Pooling, ReLU and Inner Product ones). The main reason for this choice is that the case studies are hardly different between each other but

they share the improvement of reliability following the same trend: thus the approach proposed can be fairly generic.

The experiments are made as radiation beam bombardments, following the framework described in chapter 2.

The methodology proposed by Libano *et al.* in [24] is called Selective Hardening. A very effective solution would be the Triple Modular Redundancy (TMR) applied to the whole FPGA circuit: the problem is the scarcity of resources to actually implement it, and also the significant overhead introduced by this technique in terms of area and power consumption. Here the idea is to reinforce only the layers in the network that, if corrupted, are likely causing critical errors.

Hence, a preliminary fault-injection campaign (software-based) was performed to identify the critical layers of the two networks under test. It was found that in general the first layers are more vulnerable because there is more room for a fault to propagate. Nevertheless, each network should be carefully analyzed: the sensitivity of layers depends on the topology. In the current case study the most critical layers are found as the hidden layer of the *Iris Flower ANN* and the last layer (Inner Product) of the *MNIST CNN*.

Thereafter, the radiation experiments were performed. There are four situations of chips that are compared: Unhardened (without protection), Selective TMR (protecting only the most critical layer) and Full TMR (in the cases of Hardware and Software voter). The results are presented in figures 4.4 - 4.5 - 4.6 - 4.7.

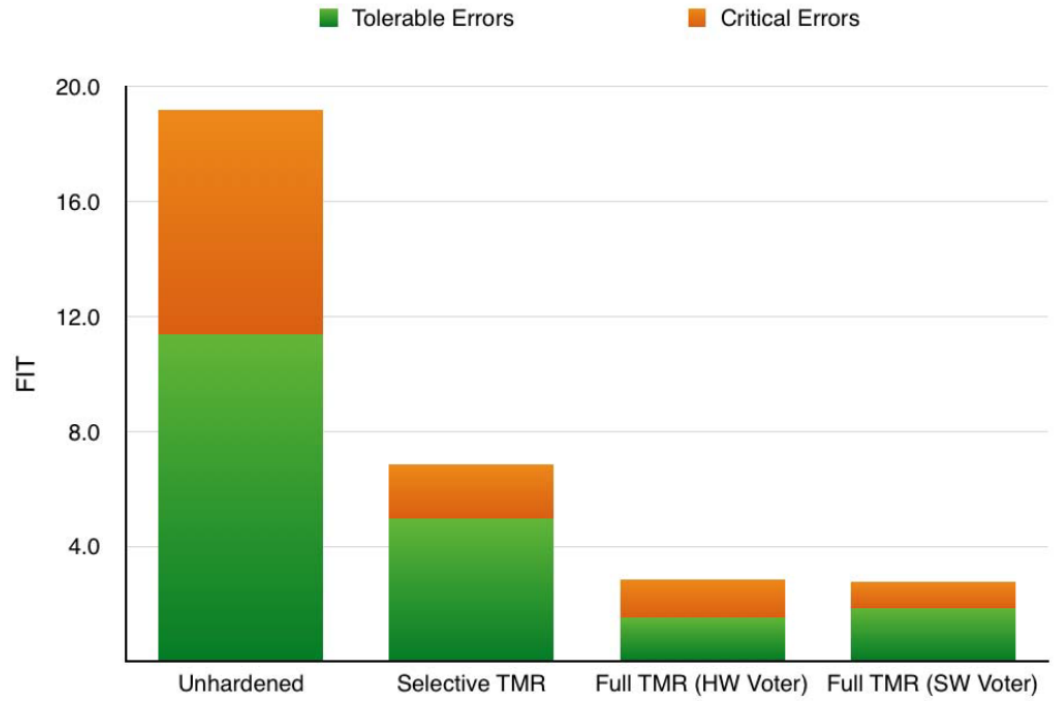

Figure 4.4. "FIT rate of the Iris Flower ANN using four different hardening configurations." [24]

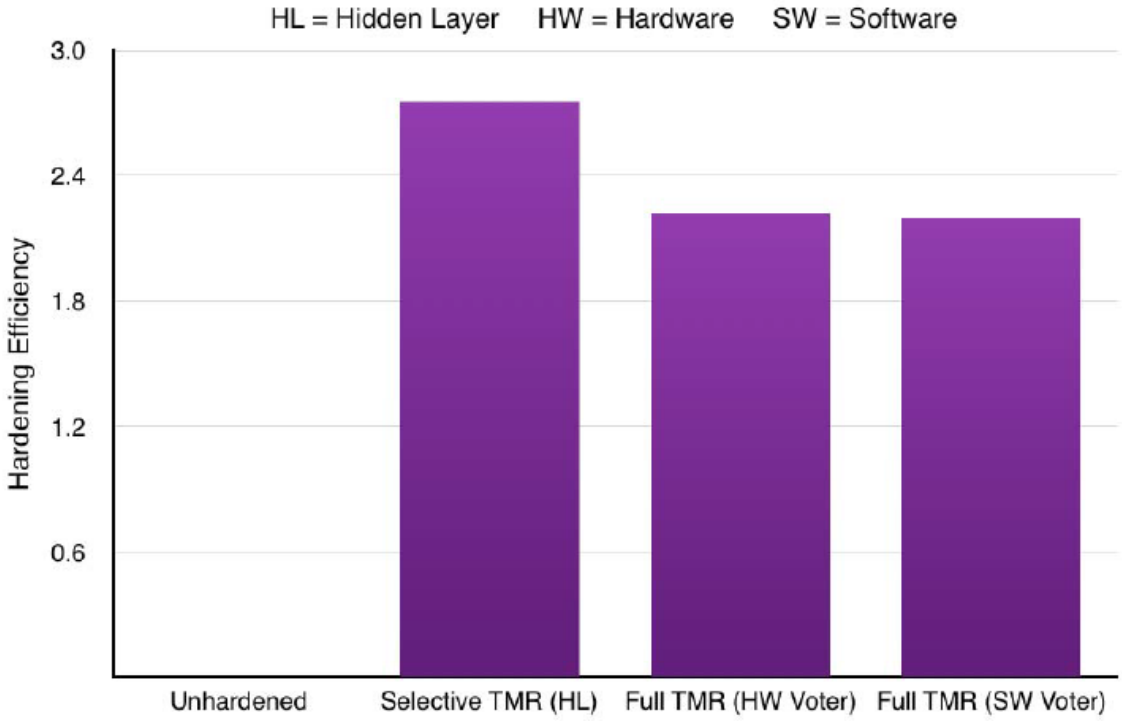

Figure 4.5. "Percentage of the masked faults divided by resource utilization of the four different hardening configurations of the Iris Flower ANN." [24]

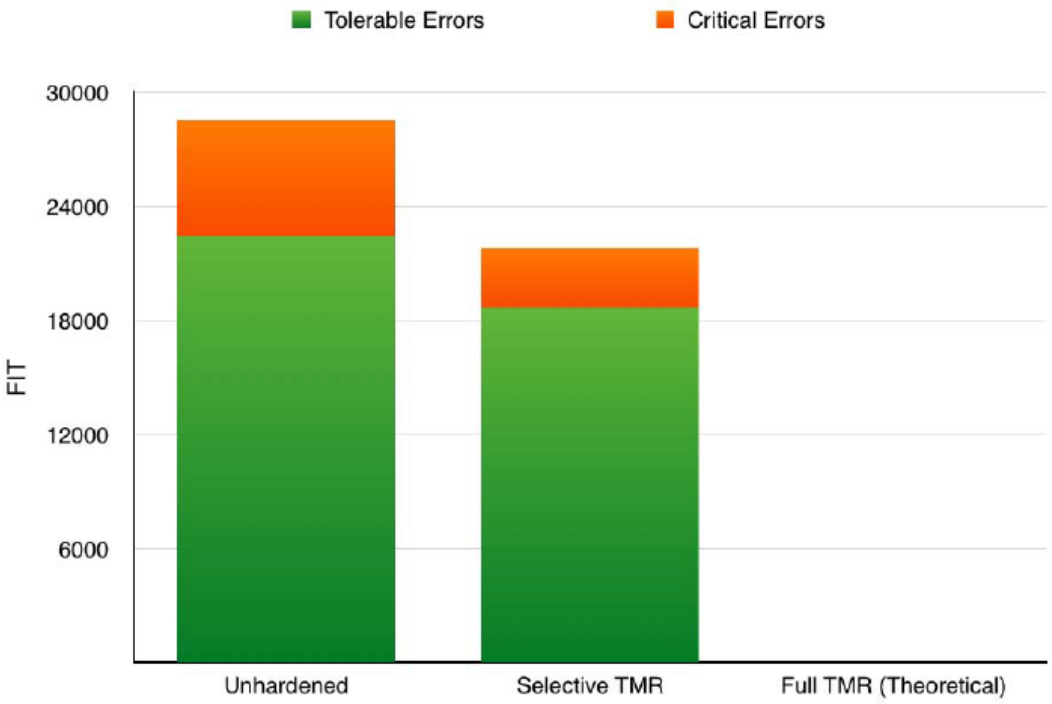

Figure 4.6. "FIT rate of the MNIST CNN using three different hardening configurations." [24]

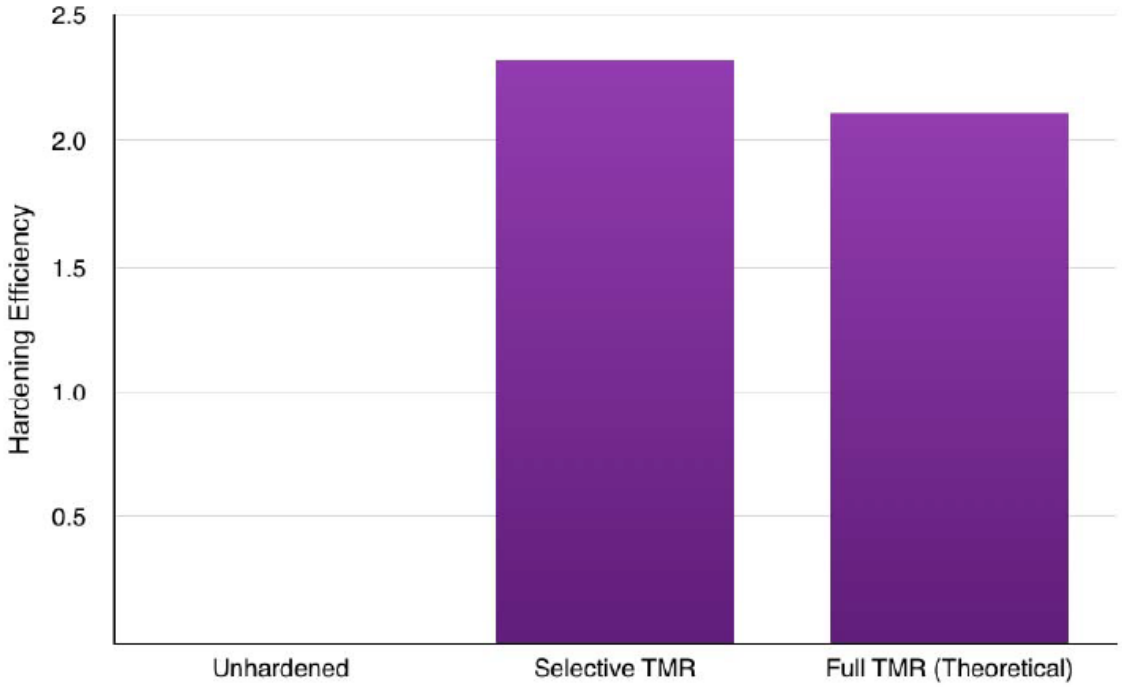

Figure 4.7. "Percentage of the masked faults divided by resource utilization of the three different hardening configurations of the MNIST CNN." [24]

Generally, it is worth noting that the majority of errors are tolerable. Then, the full TMR is more effective than the selective one in terms of percentage of errors, while it is less efficient since it needs a lot of resources to be sustained.

Hence, Selective Hardening technique is found to be a good trade off between the resource utilization and the percentage of the errors. From the graphs it is possible also to notice that it was not possible to implement the full TMR in *MNIST CNN* because of the lack of resources, and this fact emphasize the necessity of smartly protect only the most vulnerable parts of the network.
Finally, a possible improvement of this technique can be to iteratively apply this approach, protecting not only the most vulnerable layer but also the second, the third and so on, up to reach the target reliability.

4.5. Brief Observation on Aging

There exist two main aging phenomena that can have a considerable effect on the transistors of a chip: Hot Carrier Injection (HCI), mostly affecting NMOS devices, and Negative Bias Temperature Instability (NBTI), impacting mainly PMOS transistors. Both stated phenomena result in an increase of the MOS transistors' threshold voltage. This leads to a reduction of the drive current and then to an increment of the propagation delay around computation's paths. Consequently, the values whose evaluation comes from critical paths are frequently affected by aging; unfortunately, often the critical paths are used in the evaluation of significant bits in the intermediate products and this translates into considerable errors (i.e. considering the two's complement number representation, errors in higher order position can produce huge changes). Anyway, the effect of aging can be very different studying distinct instances, even when taking into account the same category of gates, but also the way gates are affected by these phenomena is related to their own stress condition (related to their actual switching activity). Another point to be evaluated is that aging is basically unavoidable, after a certain time of the operation of the chip in the field, and also that it is not possible to "reset" to its initial performances an aged transistor.
Given these facts, Liu and Chang in [25] explore the impact of circuit Aging on DNNs-based accelerators, evaluating the degradation of the accuracy that this

phenomenon is able to cause. In this study it is simulated the aging on MAC units, along days, months and years of operation and essentially in simulations they are used models for degraded MOS transistors. The neural network used here is AlexNet.

Observing the results, it is possible to provide an operative solution in order to prevent the effects on accuracy of the aging phenomenon.

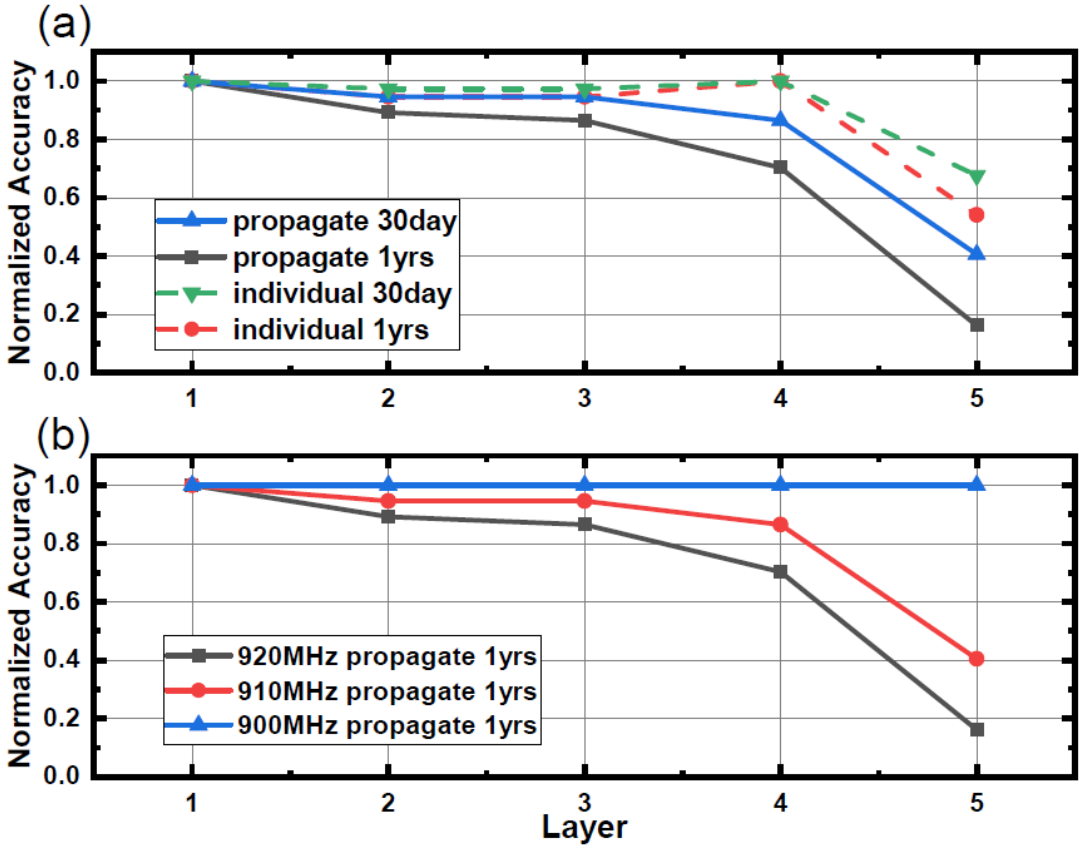

Figure 4.8. Aging simulation and evaluation [25]

In the first graph of figure 4.8, dashed lines characterize results related to single layers for which the input presented at each layer is correct and not influenced by previous stages. In this case, it is possible to notice that in the worst situation is related to the final layer, while the others' accuracy is essentially not affected. Instead, solid lines consider the situation in which it is enabled the propagation of errors, due to aging phenomena, through different layers. Here, the whole calculation suffers bigger accuracy reduction, and this can be easily reconducted to the fact that, in the considered situation, errors are propagating.

Generally, the rapid degradation of the accuracy in the final layer witnesses the fact that networks made by an high number of layers can be more vulnerable to aging phenomena. A solution comes out from the work of Liu and Chang: observing the simulation results reported in the second graph of figure 4.8, relaxing the nominal frequency of operation of the entire chip operation (920 MHz in the current case) by 10-20 MHz can significantly improve the robustness of the chip to aging phenomena. This is due to the fact that relaxing frequency there is more space for critical paths, and now they are able to mask the delay errors caused by aging.

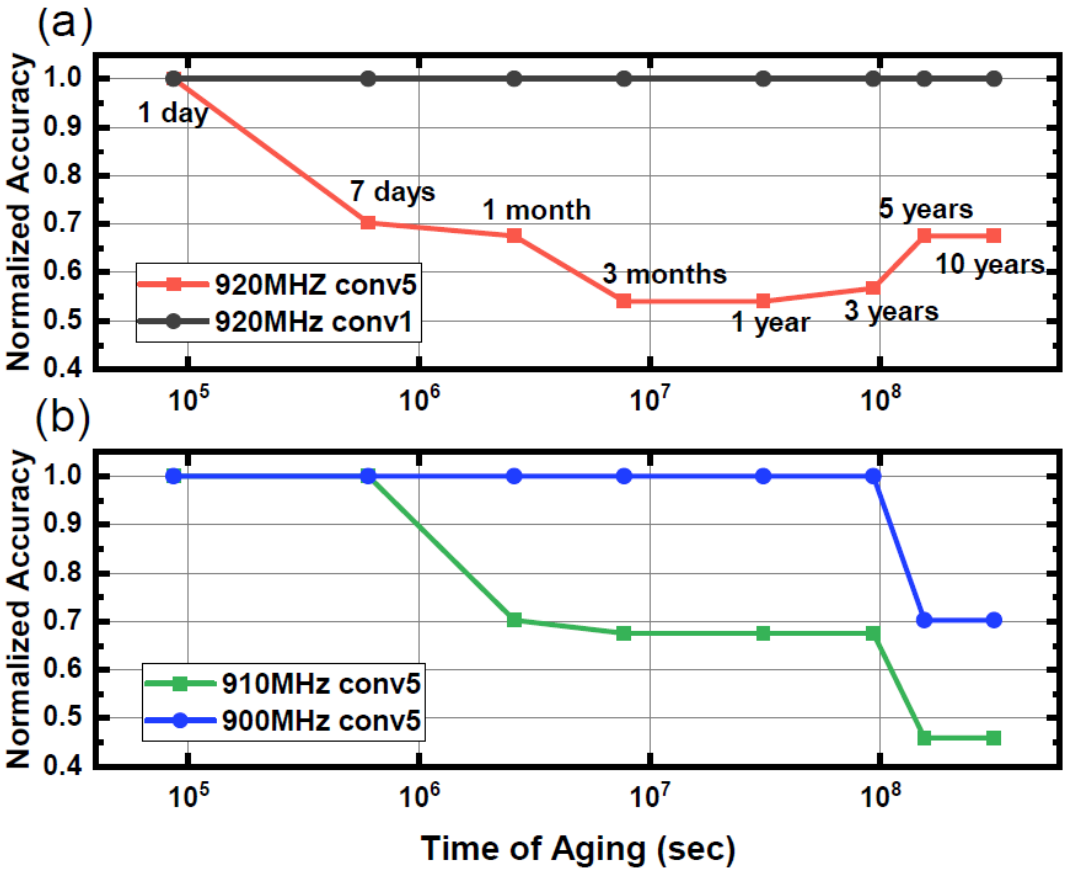

Figure 4.9. Liu and Wang simulations [25]

In the first graph of figure 4.9, there is the representation of the accuracy degradation of the first and fifth convolutional layer of AlexNet, due to aging phenomena. These results are coherent with the previous ones, stating that the final layer is suffering much more with respect to the first one. A curious

phenomenon that can be observed by this simulation is the fact that the accuracy of the fifth layer starts to recover after one years, and this is caused by the flipping of several internal signals and then it is increased the probability of accidentally correct errors. Clearly, referring to safety-critical applications, this totally unpredictable situation must definitely be avoided. In the second chart of figure 4.9, there is another confirmation of the conclusions of the study; indeed relaxing the operating frequency, chips can maintain longer their nominal performances.

Hence, device aging can result in a substantial accuracy drop after one year of deployment, and different layers have different sensitivities to this kind of phenomena.

Conclusions

This work presents an exploration of the adoption of Deep Neural Networks in Safety-Critical Systems. It was made an accurate bibliographic research about requirements that should be respected, about the main procedures to evaluate the reliability of systems to be compliant with standards, about the effects of the principal faults on Deep Neural Networks operation when implemented in different hardware architectures, and about some innovative methodologies and techniques that can be implemented to improve the reliability of such systems.

Challenges and risks related to the employment of DNNs in Computer Vision for Autonomous Driving are identified and analyzed, through a collection of state-of-the-art approaches, strategies and practices.

Acknowledgements

This work was supported by Intel Deutschland GmbH as part of the Erasmus+ Traineeship Programme. A special thank goes to the leadership of Dr. Michael Paultisch, to the helpful collaboration of Dr. Ralf Graefe, to the careful supervision of Professor Cecilia Metra and Professor Martin Omana.

Abbreviations

AI Artificial Intelligence

ABFT Algorithm Based Fault Tolerance

ASIL Automotive Safety Integrity Level

BRAM Block Random Access Memory

CNN Convolutional Neural Networks

DUE Detected Uncoverable Error

DNN Deep Neural Network

E/E Electrical and Electronic

FAP Fault-Aware Pruning

FAP+T Fault-Aware Pruning + Retraining

FIT Failures In Time

FN False Negative

FP False Positive

FPGA Field-Programmable Gate Array

GPU Graphic Processing Unit

HCI Hot Carrier Injection

IoU Intersection over Union

LFM Latent Faults Metric

LUT Lookup Table

MAC Multiply and Accumulation

ML Machine Learning

NBTI Negative Bias Temperature Instability

NN Neural Network

OOD Out-of-Distribution

PFH Probability of Dangerous Failure per Hour

QM Quality Management

SED Symptom-Based Error Detectors

SET Single Event Transient

SEU Single Event Upset

SDC Silent Data Corruption

SM Streaming Multiprocessor

SPFM Single Point Faults Metric

SRAM Static Random Access Memory

TMR Triple Modular Redundancy

TN True Negative

TP True Positive

TPU Tensor Processing Unit

WGN White Gaussian Noise

Bibliography

[1] SAE international, “U.S. Department of Transportation’s New Policy on Automated Vehicles Adopts SAE International’s Levels of Automation for Defining Driving Automation in On-Road Motor Vehicles,” *SAE Int.*, p. 1, 2016, doi: P141661.

[2] R. Debouk, “Overview of the 2nd Edition of ISO 26262: Functional Safety-Road Vehicles,” no. March, p. 13, 2018, doi: 10.13140/RG.2.2.10077.26085.

[3] J. Henriksson, M. Borg, and C. Englund, “Automotive safety and machine learning: Initial results from a study on how to adapt the ISO 26262 safety standard,” *Proc. - Int. Conf. Softw. Eng.*, pp. 47–49, 2018, doi: 10.1145/3194085.3194090.

[4] Y. Jung, S. Park, and M. Han, “Automotive Hardware Development According to ISO 26262,” *13th Int. Conf. Adv. Commun. Technol.*, pp. 588–592, 2011.

[5] P. Kafka, “The automotive standard ISO 26262, the innovative driver for enhanced safety assessment & technology for motor cars,” *Procedia Eng.*, vol. 45, pp. 2–10, 2012, doi: 10.1016/j.proeng.2012.08.112.

[6] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, “Object Detection with Deep Learning: A Review,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

[7] Y. Zhang, N. Qin, D. Huang, and K. Liang, “Fault Diagnosis of High-speed Train Bogie Based on Deep Neural Network,” *IFAC-PapersOnLine*, vol. 52, no. 24, pp. 135–139, 2019, doi: 10.1016/j.ifacol.2019.12.395.

[8] M. Henne, A. Schwaiger, K. Roscher, and G. Weiss, “Benchmarking Uncertainty Estimation Methods for Deep Learning With Safety-Related Metrics,” 2020.

[9] E. Ozen and A. Orailoglu, “Sanity-Check: Boosting the Reliability of Safety-Critical Deep Neural Network Applications,” *Proc. Asian Test Symp.*, vol. 2019-Decem, pp. 7–12, 2019, doi: 10.1109/ATS47505.2019.000-8.

[10] J. Zhang, T. Gu, K. Basu, and S. Garg, “Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator,” *Proc. IEEE VLSI Test Symp.*, vol. 2018-April, no. Ml, pp. 1–6, 2018, doi: 10.1109/VTS.2018.8368656.

[11] M. A. Hanif, F. Khalid, R. V. W. Putra, S. Rehman, and M. Shafique, “Robust Machine Learning Systems: Reliability and Security for Deep Neural Networks,” *2018 IEEE 24th Int. Symp. On-Line Test. Robust Syst. Des. IOLTS 2018*, pp. 257–260, 2018, doi: 10.1109/IOLTS.2018.8474192.

[12] F. F. Dos Santos, P. Navaux, L. Carro, and P. Rech, “Impact of reduced precision in the reliability of deep neural networks for object detection,” *Proc. Eur. Test Work.*, vol. 2019-May, no. Section III, 2019, doi: 10.1109/ETS.2019.8791554.
[13] M. A. Neggaz, I. Alouani, P. R. Lorenzo, and S. Niar, “A Reliability Study on CNNs for Critical Embedded Systems,” *Proc. - 2018 IEEE 36th Int. Conf. Comput. Des. ICCD 2018*, pp. 476–479, 2019, doi: 10.1109/ICCD.2018.00077.
[14] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, “A reliability analysis of a deep neural network,” *LATS 2019 - 20th IEEE Lat. Am. Test Symp.*, pp. 1–6, 2019, doi: 10.1109/LATW.2019.8704548.
[15] A. Lotfi *et al.*, “Resiliency of automotive object detection networks on GPU architectures,” *Proc. - Int. Test Conf.*, vol. 2019-Novem, pp. 1–9, 2019, doi: 10.1109/ITC44170.2019.9000150.
[16] P. Rech, C. Aguiar, C. Frost, and L. Carro, “An efficient and experimentally tuned software-based hardening strategy for matrix multiplication on GPUs,” *IEEE Trans. Nucl. Sci.*, vol. 60, no. 4, pp. 2797–2804, 2013, doi: 10.1109/TNS.2013.2252625.
[17] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” *Proc. - Int. Symp. Comput. Archit.*, vol. Part F1286, pp. 1–12, 2017, doi: 10.1145/3079856.3080246.
[18] H. T. Kung, “Why Systolic Architectures?,” *Computer (Long. Beach. Calif).*, vol. 15, no. 1, pp. 37–46, 1982, doi: 10.1109/MC.1982.1653825.
[19] M. A. Hanif, R. Hafiz, and M. Shafique, “Error resilience analysis for systematically employing approximate computing in convolutional neural networks,” *Proc. 2018 Des. Autom. Test Eur. Conf. Exhib. DATE 2018*, vol. 2018-Janua, pp. 913–916, 2018, doi: 10.23919/DATE.2018.8342139.
[20] K. T. Chitty-Venkata and A. Somani, “Impact of Structural Faults on Neural Network Performance,” *2019 IEEE 30th Int. Conf. Appl. Syst. Archit. Process.*, vol. 2160–052X, pp. 35–35, 2019, doi: 10.1109/asap.2019.00-38.
[21] G. Li *et al.*, “Understanding error propagation in Deep Learning Neural Network (DNN) accelerators and applications,” *Proc. Int. Conf. High Perform. Comput. Networking, Storage Anal. SC 2017*, 2017, doi: 10.1145/3126908.3126964.
[22] K. Huang and J. A. Abraham, “Algorithm-Based Fault Tolerance for Matnx Operations,” vol. c, no. 6, pp. 518–528, 1984.
[23] F. F. Dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, “Evaluation and Mitigation of Soft-Errors in Neural Network-Based Object Detection in Three GPU Architectures,” *Proc. - 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks Work. DSN-W 2017*, pp. 169–176, 2017, doi: 10.1109/DSN-W.2017.47.
[24] F. Libano *et al.*, “Selective hardening for neural networks in FPGAs,” *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 216–222, 2019, doi: 10.1109/TNS.2018.2884460.
[25] W. Liu and C. H. Chang, “Analysis of circuit aging on accuracy degradation of deep neural network accelerator,” *Proc. - IEEE Int. Symp. Circuits Syst.*, vol. 2019-May, 2019, doi: 10.1109/ISCAS.2019.8702226.