UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

FABIANO PEREIRA LIBANO

Reliability Analysis of Neural Networks in FPGAs

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Microelectronics

Advisor: Prof. Dr. Paolo Rech

Porto Alegre
August 2018

CIP — CATALOGING-IN-PUBLICATION

Libano, Fabiano Pereira

Reliability Analysis of
Neural Networks in FPGAs / Fabiano Pereira Libano. – Porto Alegre: PGMICRO da UFRGS, 2018.

72 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2018. Advisor: Paolo Rech.

1. Reliability. 2. Neural Networks. 3. FPGAs. 4. Artificial Intelligence. 5. Safety-Critical Applications. I. Rech, Paolo. II. Título.

”You have to make the rules,

not follow them.”

— Isaac Newton

ACKNOWLEDGMENTS

I would like to thank my parents Fausto Bastos Libano and Fatima Rosane Pereira Libano as well as my brother Felipe Pereira Libano, for the support not only during the last five and a half years, but throughout my entire life.

To all the professors I had, who put legitimate effort into teaching their classes and contributed to me by exposing their knowledge.

To my classmates and my fellow research colleagues (in Brazil, in the United States and in the United Kingdom) who helped me to succeed academically in a variety of aspects.

To UFRGS, Institute of Informatics and School of Engineering, for helping me to develop the most important trait of an engineer: the ability to solve problems.

To my advisor Paolo Rech who taught me what computing really is from day one, and continues to do so on a daily basis.

To all of you, I sincerely say: thank you.

ABSTRACT

Neural networks are becoming an attractive solution for automatizing vehicles in the automotive, military, and aerospace markets. All of these applications are safety-critical and, thus, must have reliability as one of the main constraints. Thanks to their low-cost, low power-consumption, and flexibility, Field-Programmable Gate Arrays (FPGAs) are among the most promising devices to implement neural networks. Unfortunately, FPGAs are also known to be susceptible to faults induced by ionizing particles. In this work, we evaluate the effects of radiation-induced errors in the outputs of two neural networks (Iris Flower and MNIST), implemented in SRAM-based FPGAs. In particular, through accelerated neutron beam experiments, we notice that radiation can induce errors that modify the output of the network with or without affecting the neural network's functionality. We call the former critical errors and the latter tolerable errors. We explore aspects of the neural networks that can have impacts on both performance and reliability, such as levels of data precision and different methods of implementation for some types of layers. Through exhaustive fault-injection campaigns, we identify the portions of Iris Flower and MNIST implementations on FPGAs that are more likely, once corrupted, to generate a critical or a tolerable error. Based on this analysis, we propose Algorithm-Based Fault Tolerance (ABFT) strategies for certain layers in the networks, as well as a selective hardening strategy that triplicates only the most vulnerable layers of the neural network. We validate these hardening approaches with neutron radiation testing, and see that our selective hardening solution

Keywords: Reliability. Neural Networks. FPGAs. Artificial Intelligence. Safety-Critical Applications.

Análise de Confiabilidade de Redes Neurais em FPGAs

RESUMO

Redes neurais estão se tornando soluções atrativas para a automação de veículos nos mercados automotivo, militar e aeroespacial. Todas essas aplicações são de segurança crítica e, portanto, precisam ter a confiabilidade como um dos principais requisitos. Graças ao baixo custo, baixo consumo de energia, e flexibilidade, FPGAs estão entre os dispositivos mais promissores para implementar redes neurais. Entretanto, FPGAs também são conhecidas por sua susceptibilidade à falhas induzidas por partículas ionizadas. Neste trabalho, nós avaliamos os efeitos de erros induzios por radiação nas saídas de duas redes neurais (Iris Flower e MNIST), implementadas em FPGAs baseadas em SRAM. Em particular, via experimentos com feixe acelerado de nêutrons, nós percebemos que a radiação pode induzir erros que modificam a saída da rede afetando ou não a corretude funcional da mesma. Chamamos o primeiro caso de erro crítico e o segundo de error tolerável. Nós exploramos aspectos das redes neurais que podem impactar tanto seu desempenho quanto sua confiabilidade, tais como os níveis de precisão na representação dos dados e diferentes métodos de implementação de alguns tipos de camadas. Usando campanhas exaustivas de injeção de falhas, nós identificamos porções das implementações da Iris Flower e da MNIST em FPGAs que são mais prováveis de gerar erros critícos ou toleráveis, quando corrompidas. Baseado nessa análise, nós propusemos estratégias de ABFT para algumas camadas das redes, bem como uma estratégia de proteção seletiva que triplica somente as camadas mais vulneráveis das redes neurais. Nós validamos essas estratégias de proteção usando testes de radiação com nêutrons, a vemos que nossa solução de proteção seletiva conseguiu mascarar 68% das falhas na Iris Flower com um custo adicional de 45%, e 40% das falhas na MNIST com um custo adicional de 8%.

Palavras-chave: Confiabilidade, Redes Neurais, FPGAs, Inteligência Artificial, Aplicações de Segurança-Crítica.

LIST OF ABBREVIATIONS AND ACRONYMS

ABFT	Algorithm-Based Fault Tolerance.
AI	Artificial Intelligence.
ANN	Artificial Neural Network.
AVF	Architectural Vulnerability Factor.
BAIR	Berkeley Artificial Intelligence and Robotics.
BNN	Binary Neural Network.
CLB	Configurable Logic Block.
CNN	Convolutional Neural Network.
DSP	Digital Signal Processor.
DUT	Design Under Test.
FF	Flip-Flop.
FPGA	Field-Programmable Gate Array.
FSM	Finite State Machine.
GPU	Graphics Processing Unit.
HDL	Hardware Description Language.
HLS	High-Level Synthesis.
IC	Integrated Circuit.
LANL	Los Alamos National Laboratory.
LANSCE	Los Alamos National Science Center.
LUT	Look-Up Tables.
MEBF	Mean Executions Between Failures.
ML	Machine Learning.
MLP	Multi Layer Perceptron.
MNIST	Modified National Institute of Standards and Technology.
PL	Programmable Logic.
PS	Processing System.

RAL	Rutherford Appleton Laboratory.
ReLU	Rectified Linear Unit.
SDC	Silent Data Corruption.
SEE	Single-Event Effect.
SEFI	Single-Event Functional Interrupt.
SEU	Single-Event Upset.
SLP	Single Layer Perceptron.
SoC	System-on-Chip.
TMR	Triple Modular Redundancy.
TRE	Tolerated Relative Error.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Feed-forward neural networks are computational approaches that have increasingly been adopted in many fields, including pattern recognition, high performance computing, and data mining (AMIN et al., 2013). Additionally, Artificial Neural Networks (ANNs) and, more specifically, Convolutional Neural Networks (CNNs) are extremely attractive for safety critical applications, such as space exploration (ALLEN et al., 2017), autonomous driving (NEAGOE et al., 2012), and military applications like UAVs.

These safety-critical applications utilize a number of pattern recognition algorithms to detect and classify objects based on captured frames or signals. Such algorithms can be implemented using ANNs and CNNs. However, in recent years researches have focused more on performance aspects of the neural networks (such as execution time and accuracy), and ended up neglecting reliability factors (such as error rate). As we know, failures in safety-critical applications can lead to loss of valuable assets, or even worse, loss of human lives. Thus, we need to properly evaluate how ANNs behave in harsh environments, and how we can apply hardening techniques to improve reliability.

Due to the intrinsic parallelism between neurons and layers, neural networks can be efficiently implemented on FPGAs (HE et al., 2009). Thanks to their low cost, low power consumption, and flexibility, FPGAs are, in fact, an attractive solution for classification tasks in safety-critical applications. Unfortunately, FPGAs have been shown to have a high radiation sensitivity (WIRTHLIN, 2015). In particular, FPGAs may experience upsets in the configuration memory that can change routing connections, or the content of Look-Up Tables (LUT), ultimately modifying the large scale implemented design. The promising aspect is that, as we explain in the following chapters, not every output error is critical for neural networks running classification tasks.

We chose to evaluate two feed-forward ANNs as case-studies (Iris Flower and MNIST), because we wanted to see if the aspects of reliability such as error criticality and layer vulnerability are somewhat general across most feed-forward ANNs. The similarity between our chosen neural networks is that they both perform classification tasks, and the differences are on size (number of layers), and also types of layers. In fact, the larger neural network (MNIST) is a CNN, and so it adds a few extra layers for feature extraction while Iris Flower receives already computed features as inputs and simply performs classification. We also chose two FPGA devices (Zynq-7000 and Zynq Ultrascale+) with different characteristics to run such neural networks. Since the resource requirements for each ANN is very different (due to their difference in size, and consequently computational complexity), we opted for an entry-level, low-cost device (Zynq-7000) to implement the smaller Iris Flower neural network, and opted for a high-performance device (Zynq Ultrascale+) to implement the larger MNIST CNN. We should also point out that the chosen devices use different technologies: 28nm for the Zynq-7000 and 16nm FinFET for the Zynq-Ultrascale+. The different transistor layout is known to bias the raw resources error rate.

The main scope of this work is to evaluate and propose efficient and effective strategies to improve the reliability of neural networks as implemented in FPGAs. Our goal in testing the Iris Flower and MNIST is to first measure their error rate and to observe how radiation can impact their execution. We found, through accelerated neutron beam experiments, that most of the radiation induced faults that propagate to the output can be considered as tolerable, in the sense that the classification task being performed is not compromised by small disturbances in the network's response. Then, to better understand error propagation within the network, we perform fault injection campaigns on both case-studies. We learn that each layer has a different level of vulnerability to radiation, suggesting that they should be treated differently when thinking of hardening techniques.

Based on the information gathered, we design new, fault tolerant versions of our neural networks. First, we tested well-known ABFT techniques applied to specific layers of the MNIST CNN, and then we propose a hardening approach that focuses on selective modular redundancy, triplicating only the most vulnerable layer in each model. Finally, we validate our hardening approach by conduction another round of beam experiments. We find that our selective hardening strategy proved to be up to 20% more efficient than traditional solutions, such as full TMR.

The remainder of this work is organized as follows. Chapter 2 provides a background on safety-critical applications, discusses technical aspects of neural networks, why they are a good fit for parallel devices, and how such devices are prone to radiation-induced errors. Chapter 3 presents an overview of the training process of a neural network, and discusses the challenges of implementing a trained model on FPGAs. Chapter 4 details of our case-study devices, neural network topologies and their respective datasets. Chapter 5 explains our experimental methodology by discussing the advantages of each experiment and the complimentary aspect of the information they provide. Chapter 6 presents the first round of experimental data, evaluating error criticality, the impact of data precision and algorithm choices in ANNs, as well as vulnerability levels across different layers in neural networks. Chapter 7 discusses ways of improving the reliability of ANNs, by presenting new beam data of ABFT-protected layers, and a proposed selective hardening approach. Chapter 8 presents conclusions and paves a path for future work.

2 BACKGROUND

This chapter introduces some main concepts such as safety and reliability. Machine learning algorithms and the state-of-the-art devices required to execute them are also presented. Finally, the chapter discusses why it is important to consider radiation effects on electronic devices and applications, and how we should classify radiation-induced errors according to the type of application we are dealing with.

2.1 Safety-Critical Applications

An application is considered safety-critical if its failure can result in loss of human lives, harm to private property or environmental damage (IEC, 2017). Systems to be used in such applications should have the following properties: reliability, maintainability, availability, safety, and security. This work focuses on reliability, which is the probability of a failure to occur, and safety, which is the probability that the system will not cause any harm in case of malfunction (IEEE, 1990). Further properties are defined by (BERNARDESCHI et al., 2015). In order to reach functional safety, a given system must be compliant with the appropriate safety constraints by knowing how to handle failures, ultimately reaching tolerable levels. These constraints are defined in international standards to help designers on the development and the qualification of reliable systems. One of the most actual and important examples of safety-critical applications is the fast-growing autonomous trend in the automotive sector. Every year, the level of automation in partially self-driving cars rises, and we will inevitably reach fully autonomous conduction, sooner rather than later (ELECTREK, 2016). It is clear that a self-driving car must have reliability as one of the primary concerns. The military and aerospace markets are also massively dependent on reliability and use rad-hard components or apply a number of fault tolerance techniques to guarantee missions functional safety. This is mandatory as billions of dollars are at stake, not to mention human lives. Unfortunately, the use of rad-hard devices is not ideal for automotive applications as they would significantly increase project costs. Moreover, as discussed in the following chapters, the adoption of traditional hardening solutions like full triplication may be unfeasible in real-time systems.

The safety-critical area is, of course, highly regulated with norms such as the IEC 61508 and ISO 26262, which restricts the set of solutions that can be commercially adopted, but do not forbid scientific research: they rather stimulate it.

Figure 2.1: Safety-critical applications

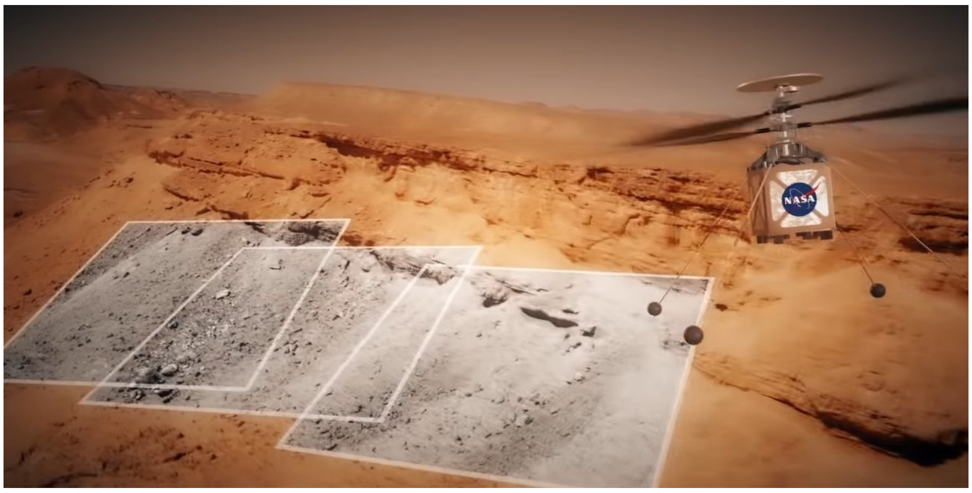
(a) Tesla's autopilot function

Source: (ELECTREK, 2016)

(b) U.S. Air Force drone

Source: (MILITARY.COM, 2015)

(c) NASA's upcoming autonomous helicopter



Source: (VALUEWALK, 2018)

2.2 Artificial Intelligence, Machine Learning

In computer science, the Artificial Intelligence (AI) field is dedicated to the study of intelligent agents, that are able to perceive the surrounding environment, make decisions, and take actions to maximize their chance of successfully achieving their goals. The term AI is also used to describe machines that are able to mimic cognitive human capabilities (RUSSELL; NORVIG, 2010). One of the main subareas of AI is Machine Learning (ML), which consists in a set of algorithms and methods that allow a computer to learn without being explicitly programmed (SAMUEL, 1959). Generally speaking, ML adoption is highly recommended in applications where the construction of a deterministic algorithm is very time consuming, such as computer vision and predictive systems. The actual learning process involves several hours of supervised or unsupervised training, where the system is fed several times with large datasets, and adjusts its own behavior after each iteration. Essentially, ML algorithms learn from input patterns, and eventually become able to spot details, characteristics, and patterns that humans are not able to deterministically describe. There are a number of approaches within the ML field, such as Decision Trees, Bayesian Networks, Reinforcement Learning, Genetic Algorithms, and many others. This work explores ANNs, and their biologically inspired structures of interconnected neurons, working together to model complex relationships between inputs and outputs. Recently, the research on ANNs has increased significantly, with large scale topologies (surpassing thousands of neurons) and Big Data usage in the training processes (Chapter 3.1 dives deeper into them). Also, efforts towards the construction of dedicated Integrated Circuit (IC) have been made by big market players, such as IBM with its TrueNorth chip, capable of simulating up to 256 million synapses in real-time (MEROLLA et al., 2014). In addition, neural networks are consistently proving to perform better than human professionals in a number of specific fields, which can be exemplified by Google's DeepMind AlphaGo beating the world's best player of Go (millenary Chinese table game) (QUARTZ, 2016).

2.3 Feed-Forward Artificial Neural Networks

As previously stated, ANNs are inspired by the way our biological brain works. Sets of interconnected neurons, propagating signals like synapses, in a highly parallel fashion, to accomplish complex tasks (from a macro perspective).

One of the very first successful types of ANNs was based around the Perceptron

(ROSENBLATT, 1958), which is an algorithm for supervised learning. First, the Single Layer Perceptron (SLP) was introduced, being a structure capable of classifying linearly separable cases (i.e., we can draw a line in an Euclidean plane to separate two sets of points). SLP does so by performing a sum of weighted inputs and then comparing the result to an activation threshold, to ultimately produce an output of 0 (if under the threshold) or 1 (if over the threshold). In order to expand the mathematical modeling capability of the SLP, the Multi Layer Perceptron (MLP) was introduced (MINSKY; PAPERT; LEON, 1969). MLP added the notion of *layers* of neurons (i.e., a pipeline of connected SLP), as well as the idea of non-linearity in activation functions (for the ability to calculate more than just linear combinations). We can see an example of a very simple ANN in Fig. 2.2. It has only three layers, named *Input Layer*, *Hidden Layer* and *Output Layer*. We can clearly perceive that every neuron in a given layer N is connected to every neuron from layers N-1 and N+1. We call this property *full connection*. Beyond that, we can see the internal structure of a neuron in Fig. 2.3. Each neuron performs a weighted sum of its inputs, and applies a non-linear activation function to the intermediate result. In the past, the most common activation functions were the sigmoids, like the hiperbolic tangent and the logistic function (illustrated in Fig. 2.4).

Figure 2.2: A simple Multilayer Perceptron

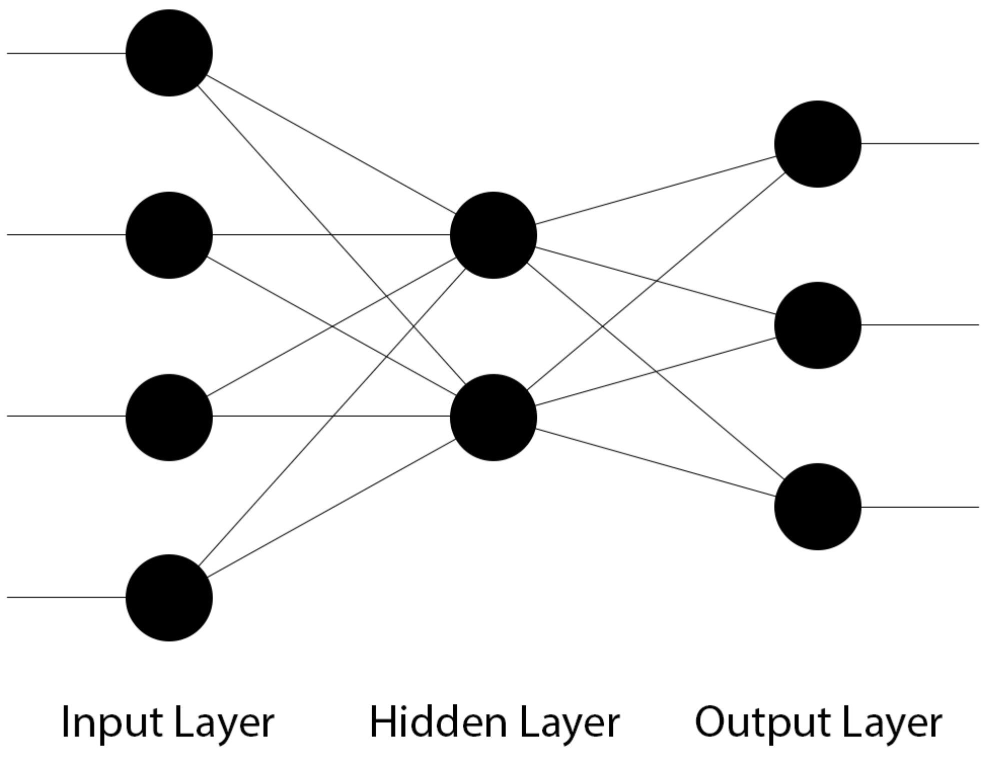


Figure 2.3: The internal structure of a neuron

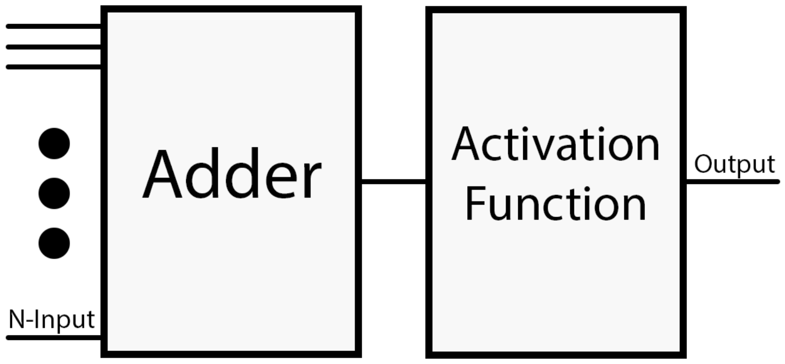
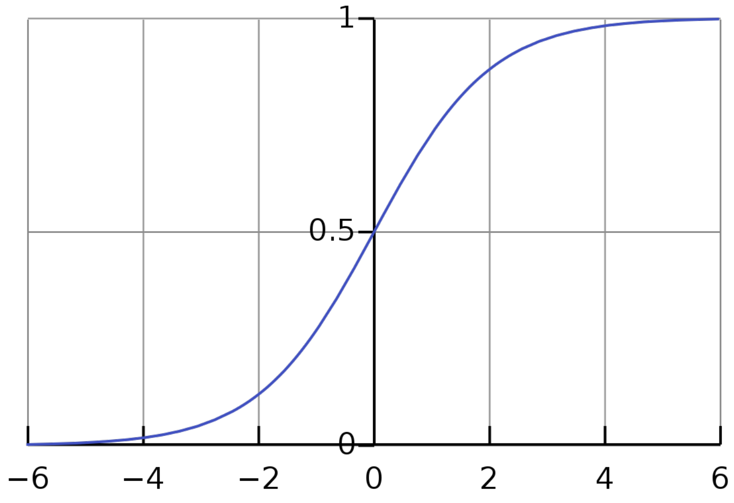


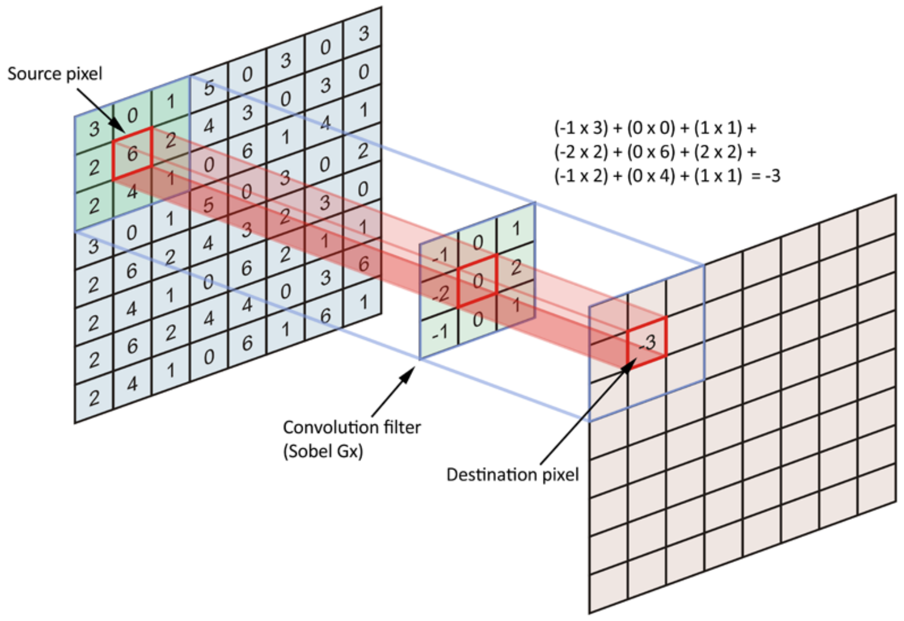
Figure 2.4: The sigmoidal logistic function



As neural networks became more popular, they started to be utilized in increasingly more complex tasks. This led to two main problems with MLPs: due to the fully-connected nature of the network, adding neurons to mimic more complex functions, rapidly increases the number of arithmetic operations (meaning that we need more computing power to execute the ANNs), and also increases the number of learnable weights (meaning that the time necessary to train the neural network, as well as the amount of memory required to stored the weights, rises considerably). To overcome these issues, researches started looking for ways to explore spatial and temporal invariance in recognition tasks. For instance, if we consider an image as an input for a MLP classifier, we would have to connect every pixel to every neuron in the following layer, performing a lot of unnecessary operations. Furthermore, our object of interest, say a "car" or a "pedestrian", could be anywhere on that image, which means that the group of neurons receiving inputs from the lower left corner of the image will have to learn to represent "pedestrian" independently from the group of neurons connected to upper right corner.

To solve these problems and disclose efficient and accurate object detection and classification, *Convolutional* Neural Networks (CNNs) were introduced (LECUN et al., 1998). The basic idea behind CNNs is to incorporate a feature extraction phase before using a MLP. The feature extraction has the role to extract specific and essential information from the input. On images, for example, this information can include shapes, shadows, edges, and others. To perform feature extraction, two new types of layers were introduced: the *convolutional layer* and the *pooling layer*. Convolutional layers apply filters (arrays of learnable weights) that slide over the input image, from the top left to the bottom right corner, executing element-wise multiplications, and then accumulating the results for each position of the filter (that moves at a predetermined rate called "stride"). Each filter in a convolutional layer is responsible for extracting a feature from the input. The first layer receives the raw image as an input while the others receive the output of the previous layer as input. As we increase the number of convolutional layers, we also increase the number of extracted features. Depending on the application, state-of-the-art CNNs can have tens of convolutional layers. To illustrate the convolution process let's consider an input image of 4x4, a filter of size 2x2 and a stride of 1. This means that the filter can move to 3 different positions in the x-axis, and 3 different positions in the y-axis (9 in total), and for each position, 4 (2x2) multiplications, followed by 1 sum of 4 elements, will be executed, constructing a 3x3 output image. Fig. 2.5 illustrates with another example.

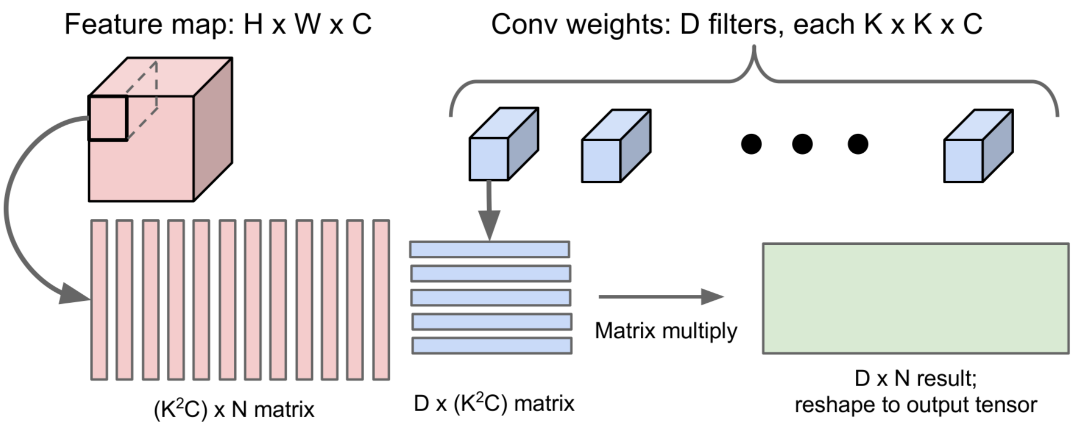
Figure 2.5: The operations in a convolutional layer


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Source: (DATA SCIENCE-STACK EXCHANGE, 2017)

Note that the convolution operation can also be modeled as a matrix multiplication (CHELLAPILLA; PURI; SIMARD, 2006), as illustrated by Fig. 2.6. On computing devices like parallel CPU or Graphics Processing Units (GPUs) it is common to implement convolution as matrix multiplication to improve performances. On FPGAs, on the contrary, traditional convolution seems to be the most effective implementation. In Chapters 6 and 7 we discuss possible advantages and disadvantages of implementing these designs in FPGAs.

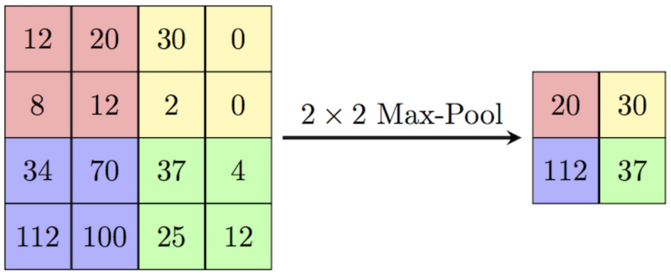
Figure 2.6: Convolution as matrix multiplication



Source: (ABRACADABRA, 2017)

Pooling layers are the other fundamental kind of layers in CNNs. They are used to progressively reduce the spatial size of the data, consequently reducing the amount of parameters and computation in the network. In fact, we can see a funnel-like structure in CNNs, as we move through the layers, extracting information. Pooling layers use the same idea of sliding filters all across the input image, except there are no learnable weights here. The most common types of operations in pooling layers are the *maximum* and the *average*. The former propagates only the element with the highest value in the considered region while the latter propagates the average value of the elements in the region. For instance, consider a 4x4 input image, a 2x2 filter, a stride of 2, and the maximum as the pooling operation. With these parameters, the filter can move to 2 different positions in the x-axis, and 2 different positions in the y-axis (4 in total), and for each position, we compare a set of 4 elements to find out the maximum value among them. Fig. 2.7 illustrates this description.

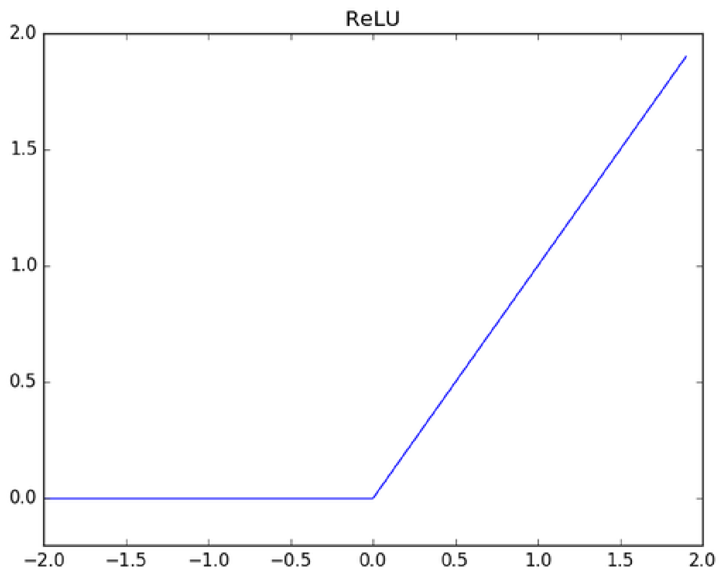
Figure 2.7: The operations in a pooling layer



Source: (CAMBRIDGE SPARK, 2017)

After the feature extraction process is complete, CNNs work the same as MLPs. In fact, in the MLP part of state-of-the-art CNNs (the last few, full connected layers), the chosen activation function is the Rectified Linear Unit (ReLU) (HAHNLOSER et al., 2000), instead of the sigmoids. This is because ReLU accomplishes the same goal, while being more computationally efficient (only a comparison is needed), and better at the training phase, as it encounters fewer vanishing gradient problems (GLOROT; BORDES; BENGIO, 2011). Fig. 2.8 showcases the ReLU function.

Figure 2.8: The ReLU activation function



We have seen that neural networks are intrinsically highly parallel structures. We have also seen that they generally require a lot of computing power, due to the high number of arithmetic operations executed in each of the layers, and a lot of memory due to the high number of weights. In Chapter 2.4, we are going to discuss which types of devices can deliver the performance required by state-of-the-art ANNs.

When considering the reliability of ANN it is also fundamental to highlight that the classification tasks have a hint of approximate computing. Since the last layer of a classifier ANN outputs the probability of each object class, there is a set of output values that result in the same classification/detection. This means that even if the actual probabilities are corrupted, the network could still produce a correct value, as long as the probability order doesn't change (meaning that the index of the highest value is still the expected one). In Chapter 2.6 we further discuss the concept of *error criticality*.

2.4 FPGAs, GPUs, SoCs

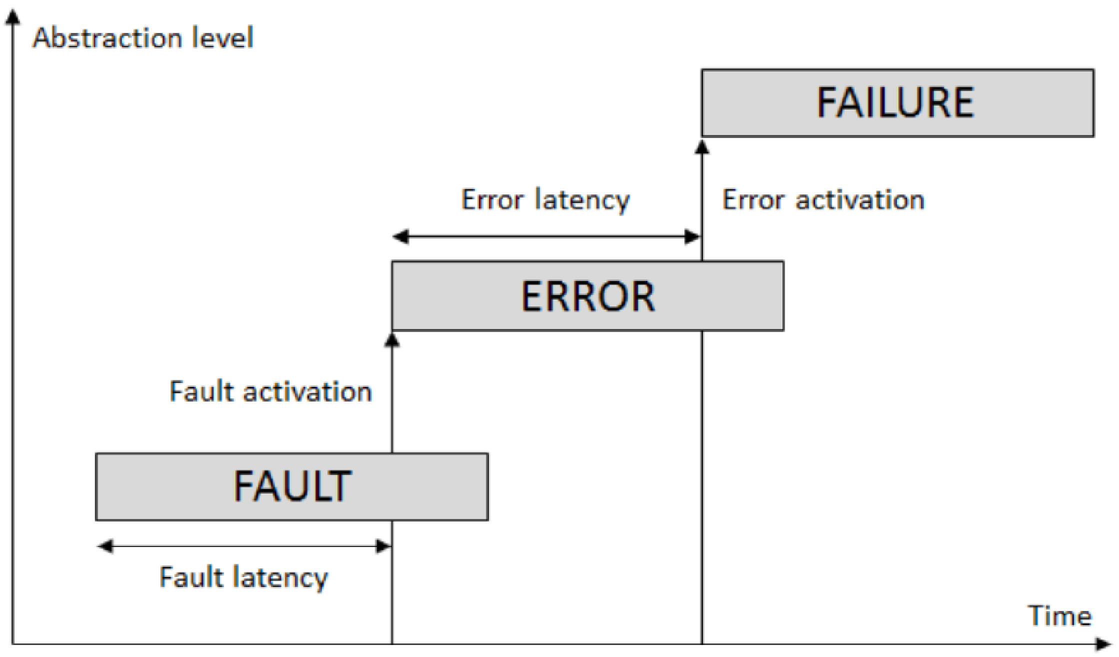
As previously stated, ANNs usually require a lot of computing power to be executed, especially if they are required to be real-time (as in the vast majority of safety-critical applications, including the automotive one). The importance of ANNs for safety critical applications and the amount of money involved in these markets drive the major hardware vendors invest in research and to produce novel devices with extremely high computing capabilities.

As Moore's law (MOORE, 1965) states, the transistor density in processors doubles every 18 to 24 months, and this has been confirmed up until the past decade. Nowadays this evolution rate seems to be slowing down. One of the most promising ways researchers designed to increase performance is the adoption of parallel processing architectures, such as FPGAs and GPUs. While GPUs usually deliver better performance, FPGAs enable, in addition to achieving lower power consumption figures, high flexibility due to their ability to implement arbitrary logic circuits using LUTs. These devices have been proven to be very suitable for parallel tasks and to deal with pipeline-friendly data flows. Today, there is also a trend of another type of heterogeneous device, that combines traditional processors with FPGAs and/or GPUs for accelerating parallel tasks. Such devices are commercially called System-on-Chips (SoCs). In the particular case of heterogeneity by FPGA usage, the devices are usually composed of two main parts: Programmable Logic (PL) and Processing System (PS), with the PL part being based on a FPGA and the PS part having single or multicore traditional processors. Finally, there are bus interfaces for communication between PS and PL, alongside optional peripherals. Several state-of-the-art SoCs are currently available, such as the Cyclone V (ALTERA, 2017) from Altera, the SmartFusion (MICROSEMI, 2017) from Microsemi and the Zynq family (XILINX, 2017) from Xilinx. This work uses as case-studies two devices of the latter, which will be presented in Chapter 4.1. Researchers have improved significantly the efficiency of computing devices. However, they may have been overly focused on performances, neglecting other critical aspects like reliability. As discussed in Chapter 2.5 the novel architectures may have several vulnerabilities that could undermine their reliability. We cannot trade-off performance for reliability in safety-critical applications, meaning that we do have to consider the coexistence of both, in order for a system to be considered optimal in its task.

2.5 Radiation Effects on Electronic Devices

According to (AVIZIENIS et al., 2004), a system is an entity composed by one or more subsystems that interact with each other, while the system itself interacts with other systems in its environment. Given this scenario, we can say that the service provided by a particular system is its behavior as perceived by its users. Thus, we can say that a system presents an error when its output is different from the expected one, and it fails whenever there its behavior deviates from the norm. Finally, we define fault, error, and failure as concepts linked by a cause-effect relationship, with an increasing level of criticality, as illustrated by Fig. 2.4.

Figure 2.9: Fault, error and failure propagation



Source: (TAMBARA, 2017)

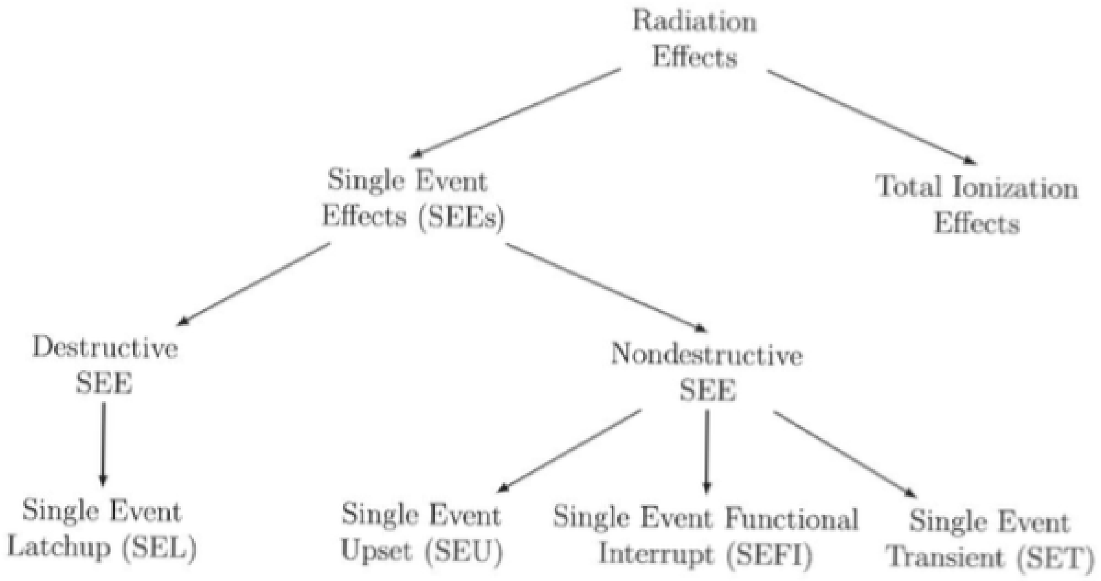
Radiation is a naturally occurring phenomenon which is caused by the transmission of energy through particles hitting the sensitive area of the device. There are many sources of radiation in the universe, such as stars (like our sun) and cosmic rays, but there are also man-made radiation rich environments like particle accelerators and nuclear reactors, for scientific purposes and energy supply, respectively. While most particles are deviated by the terrestrial magnetic field, the most energetic cosmic-rays reach the Earth. As these cosmic-rays collide with the nuclei of atoms in our atmosphere, they produce a variety of secondary particles, including alpha particles, protons, gamma, and, mainly, neutrons. The flux of neutrons that reaches ground has been measured to be of about 13 $n/(cm^2 \times h)$ in New York City. The flux changes with latitude and longitude and increases exponentially with altitude.

The advances in both architecture and fabrication processes of ICs, over the past two decades, have led to a considerable increase in transistor density (due to the reduction of transistor's size) and a reduction in power consumption (due to lower voltages). The combination of these two factors brings a greater radiation sensitivity (BAUMANN, 2005). As we previously mentioned, there is a variety of secondary particles produced after the interaction between cosmic-rays and our atmosphere, and each type of particle interacts with electronic devices in a distinct manner. We classify the particle interaction as a Single-Event Effect (SEE) when a single particle deposits enough charge in the circuit to alter its normal functionality. SEEs can still be subdivided in Destructive or Non-Destructive. Destructive SEEs are those in which either the device gets permanently damaged. Non-Destructive SEEs on the other hand, comprise the cases where a particle effects the circuit in such way that leads to error or failure in the system/application.

Whenever the particle affects the configuration memory of an FPGA, thus corrupting the bitstream, Single-Event Upsets (SEUs) have a persistent effect. The fault, in fact, will persist until a new or a corrected bitstream is loaded to the FPGA (known as memory scrubbing). If the circuit in the FPGA keeps running regardless of the fault in the bistream, the SEUs end up leading to Silent Data Corruptions (SDCs). An SDC means that the application completes the execution, but the result is not the correct/expected one. As the error is silent, we don't know whether or not the provided output is correct/trustworthy. The effect is persistent in the sense that you need to reprogram the FPGA in order to correct it, but there are hardening techniques, such as scrubbing (TONFAT et al., 2015) that work towards maintaining the original bitstream intact. Other than that, we can adopt hardening strategies at a architectural level, such as Triple Modular Redundancy (TMR)

(PRATT et al., 2008), and at the algorithm level, such as ABFT techniques (JACOBS; CIESLEWSKI; GEORGE, 2012), as an attempt to detect and correct faults, ultimately masking SDCs. Single-Event Functional Interrupts (SEFIs) manifest themselves as application crashes or device hangs. We found that SEFIs are not as common on FPGAs implementing circuits like ANNs, as they are on CPUs and GPUs. In fact, in our beam experiments, we've seen such a low quantity of SEFIs, that they became statistically irrelevant when compared to the SDCs. Fig. 2.5 shows all the main possible radiation effects.

Figure 2.10: Radiation effects


Radiation Effects
Single Event Effects (SEEs)
Total Ionization Effects
Destructive SEE
Nondestructive SEE
Single Event Latchup (SEL)
Single Event Upset (SEU)
Single Event Functional Interrupt (SEFI)
Single Event Transient (SET)

Source: (TAMBARA, 2017)

2.6 Error Criticality

A radiation-induced output error is not always critical for a neural network. This is because, as mentioned in Chapter 2.3, ANNs have a natural aspect of approximate computing, since there is usually a range of possible correct values for the output, as opposed to a purely arithmetic operation, where the expected output is a single specific value. Specially when it comes to classifiers, which is the case of the case-studies to be detailed in Chapter 4.2, a corrupted output can still lead to a correct behavior. In other words, a particle can induce the ANNs to produce wrong output values but, based on these (corrupted) values the system can still be considered properly functional (in this case, when it correctly classifies a given instance). In this situation, we consider the error as *tolerable*. Otherwise, the error is marked as *critical*. Note that we adopt this nomenclature to facilitate the understanding, since the formal definition in Chapter 2.5 might be confusing to readers who are unfamiliar with the area. In Chapters 6 and 7, we classify all of our experimental results with fault injection and radiation errors as either tolerable or critical.

Table 2.1: Definition of tolerable and critical errors

Tolerable Error	The output is different than the expected, but the **classification is still correct**.
Critical Error	The output is different than the expected, and the **classification is wrong**.

3 IMPLEMENTATION DETAILS

3.1 Training Neural Networks

In order to achieve satisfactory accuracy on a trained model of a neural network, there are a number of things we need to worry about. Everything starts with the definition of the problem we are trying to solve. For instance, if we are dealing with a classification problem (which seems to be one of the most common tasks for ANNs nowadays), we need to define the number of possible classes we want to recognize (which will be equal to the number of outputs in our model), and we also need to define the type of stimulus that the neural network is going to receive (which will determine the number of inputs in our model). We can opt to create a model where the inputs are preprocessed attributes (Perceptron), or we can choose to use raw data and additional layers for feature extraction (which is the case of the well known Convolutional Neural Networks, CNNs).

Most of the time, the decisive factor that impacts on this choice is the dataset that we are going to use for training/testing purposes. As a result of the computational processing power advancements predicted by Moore's Law, and the rapid growth of the Internet, we've seen a huge increase in data traffic and storage. Because of this scenario, a new research field, dedicated to find optimal ways to handle large quantities of data, emerged in Computer Science, and was conveniently named Big Data. Specialists in this new field are commonly called Data Scientists. As we can intuitively guess, most of the available datasets on the Internet are raw by nature, so it is the job of the data scientist to organize the data in a way that will contribute to a successful training (for example, annotating unclassified instances). The quality of the constructed dataset will directly impact the final accuracy of the trained model.

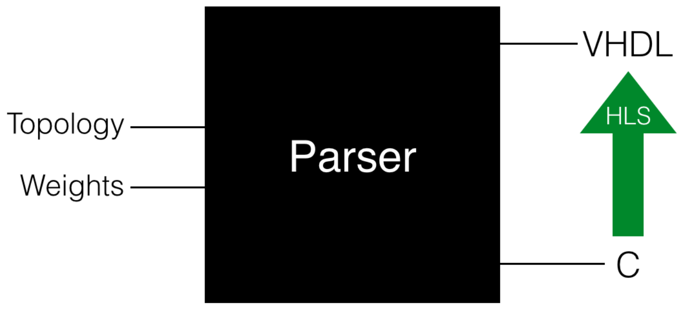
After preparing a good dataset and defining both input and output characteristics of our model, we now need to determine the middle layers (often referred to as hidden layers) and all of their parameters. To do so, we usually go back to existing state-of-the-art models that successfully deal with a similar type of data, and adopt their characteristics as a starting point. Finally, we need to define training parameters, such as how to split the available dataset for training and test phases, method of weights initialization, training function, learning rate, and many more. We iteratively go through this process, until our model converges to a desired level of accuracy.

3.2 Generating HDL Code

In last few years, to help us go through the process of training a neural network, both academia and private enterprises started to develop dedicated frameworks. We can mention two in particular: *Caffe* and *TensorFlow*. Caffe was originally developed by the Berkeley Artificial Intelligence and Robotics (BAIR) group, at University of California, Berkeley, but it was recently acquired by Facebook. TensorFlow on the other hand, started out as an internal project at Google, but was eventually made open-source, and has been massively adopted ever since. Each training framework has its own particularities, advantages and disadvantages, but they essentially accomplish the same thing: they generate a trained model (which is composed by the tuple: <topology, weights>).

If we want to run our trained model in a CPU or GPU, we just need to instantiate our trained model using our preferred programming language and feed it with new data. But in order to run it on an FPGA, we need to take a few extra steps. Basically, we new to build a parser that takes our <topology, weights> tuple as an input, and outputs Hardware Description Language (HDL) code directly or intermediary C code, which can then be used in a High-Level Synthesis (HLS) tool to generate our neural network design. This process is illustrated in Fig. 3.1.

Figure 3.1: HDL/C code generator



We should point out that generating HDL code directly is much harder (as the level of abstraction is lower), but it usually leads to better performance by using more resources and taking less clock cycles to process a given set of inputs. Our parser tool, when generating HDL, goes for a purely pipelined design, with no Finite State Machine (FSM) to control it, meaning that every single layer of the ANN is implemented on the FPGA, with pipeline stages in between them. When generating C code, the level of sequentiality/parallelism will ultimately depend on the parameters specified inside the HLS tool. As an example, the designer can opt for strategies like loop unrolling to increase parallelism in the generated design, consequently reducing the number of clock cycles taken by the ANN to process a given input, in exchange for higher resource utilization.

From our experience, we believe that the time invested towards further development of the HDL code generator doesn't pay off. The first reason is that there are a lot of nuances that the developer needs to account for, specially when dealing with CNNs, that prevents us from reaching a truly generic tool, that can handle every single ANN topology. On top of that, the generated HDL code usually requires a lot of resources in order to be implemented, so this strategy is not likely to scale for larger neural networks, as we wouldn't have FPGAs big enough to fit the designs. Despite that, there are some tools developed in academia, like *Haddoc2* (ABDELOUAHAB et al., 2017), which tries to generate HDL code from using the outputs of Caffe.

Generating C code, on the other hand, is much more feasible, as the level of abstraction is much higher. Also, considering that the HLS tools are improving year after year, we believe that this is probably the best route to take, when it comes to implementing a trained model of a neural network in FPGAs. Big companies seem to share our opinion, since they are developing proprietary tools, aiming to achieve the same outcome as we did in this work, which is to build a bridge between training frameworks and FPGAs.

Another aspect that is facilitated when using C code is that we can more easily vary the precision of data used to represent weights and pixels on the CNN. This is interesting as, due to the approximate nature of neural networks, it may not be necessary to execute operations with full precision. As demonstrated by (DING et al., 2017) even 1 bit can provide good accuracy. Our idea, as detailed in Chapter 6.1, is to evaluate how a reduced precision in the operations impacts the neural network reliability. In fact, while being more efficient, low-precision operations may be more vulnerable: A fault in a 64 bits wide number is much less likely to significantly change the output than a fault in a 32 or 16 bits number. But, at the same time, a 64 bits wide circuit occupies much more area than a 32 or 16 bits circuit, so it has more area exposed to the radiation. We can see that there is no straight-forward answer here, thus we do need to evaluate the designs experimentally.

As pointed out, we can use 64, 32 or 16 bits with floating point (double, single and half precision, respectively), but we can also go as far as using binary weights, on the so called Binary Neural Network (BNN), for example (ZHOU; REDKAR; HUANG, 2017). Though, we need to keep in mind that reducing the precision of a trained model might affect its accuracy, so whenever the designer does decide to reduce precision, he needs to go through another round of testing with the neural network, and eventually have to account for reduced precision during the training phase as well.

4 CASE STUDIES

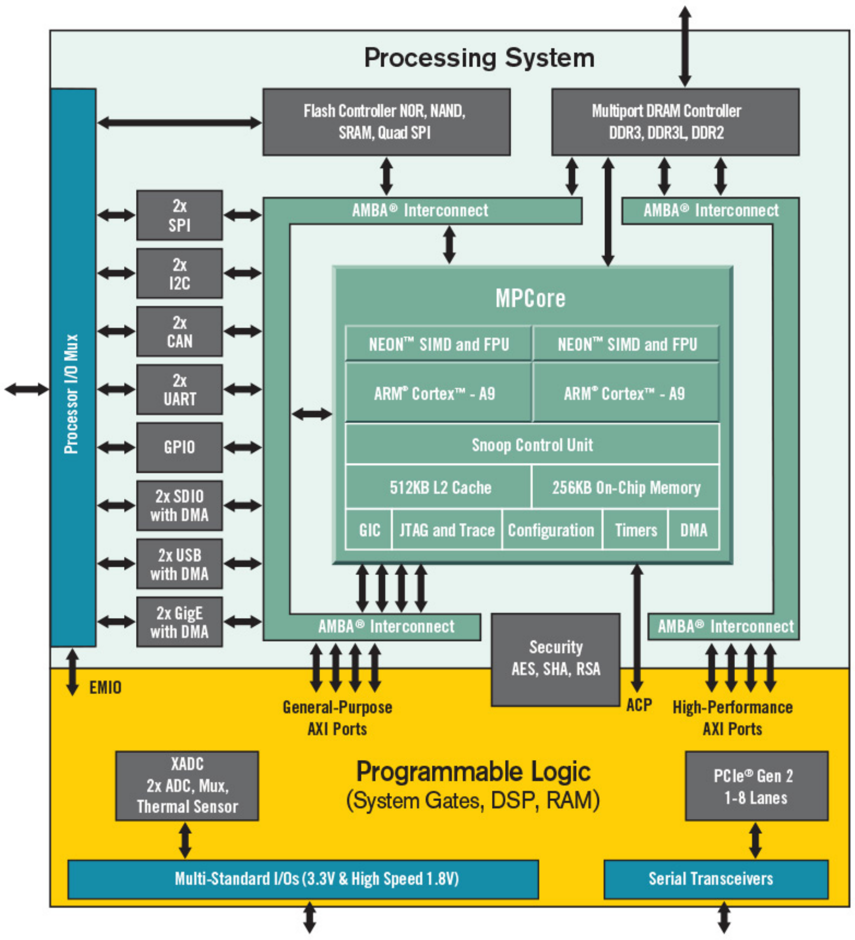
This chapter presents the Zynq-7000 APSoC and the Zynq Ultrascale+ devices. Then, it details the two chosen datasets and the correspondent ANN topologies, with resource utilization of all tested models statistics.

4.1 Devices

4.1.1 Zynq-7000

The Zynq-7000 is a device designed by Xilinx using a 28nm technology. In specific, this work utilizes the XC7Z020-CLG484 part, which is commercially available in the ZedBoard Development Board. Fig. 4.1 shows its block diagram. As we can see

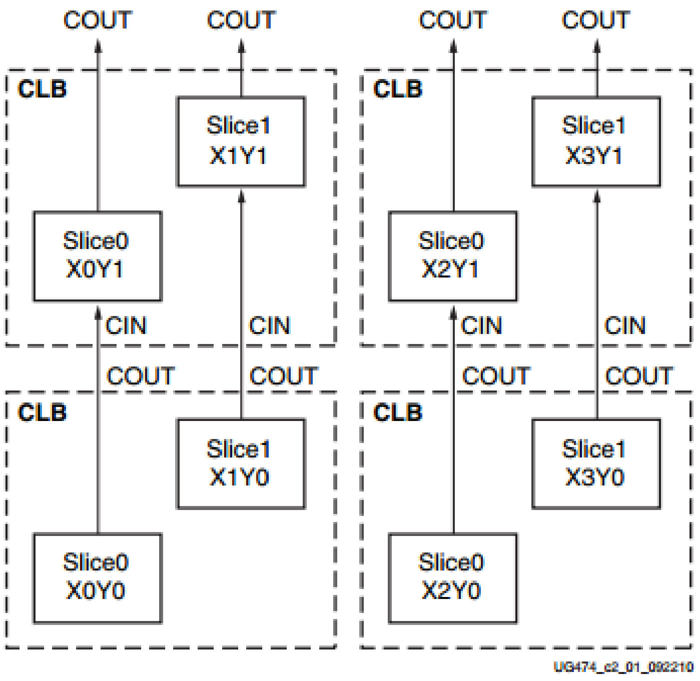
Figure 4.1: Zynq-7000's block diagram



Source: (XILINX, 2018a)

from Fig. 4.1, there are two main parts in the Zynq-7000, which are the PS and the PL, and they are connected mainly through Advanced eXtensible Interface (AXI) ports. The PS has a dual core ARM Cortex-A9 processor alongside other resources such as Floating Point Units (FPUs) and Memory Management Units (MMUs). The PL part of the Zynq-7000 has the same internal architecture as two other families of Xilinx FPGAs: Artix-7 and Kintex-7. This architecture is composed mainly of Configurable Logic Block (CLB), Digital Signal Processor (DSP) blocks and embedded memory blocks (BRAM). A set of programmable interconnections creates an array of programmable logic blocks of different types. All of these components are configured by the bitstream file, which loads into the configuration memory during the device power-up, ultimately defining a circuit previously described in a HDL. CLBs are used to implement the logic of the user's design. Each CLB is composed of one or more slices, and a slice contains one or more LUTs, Flip-Flops (FFs), and routing structures. Fig 4.2 illustrates the relationship between CLBs and slices.

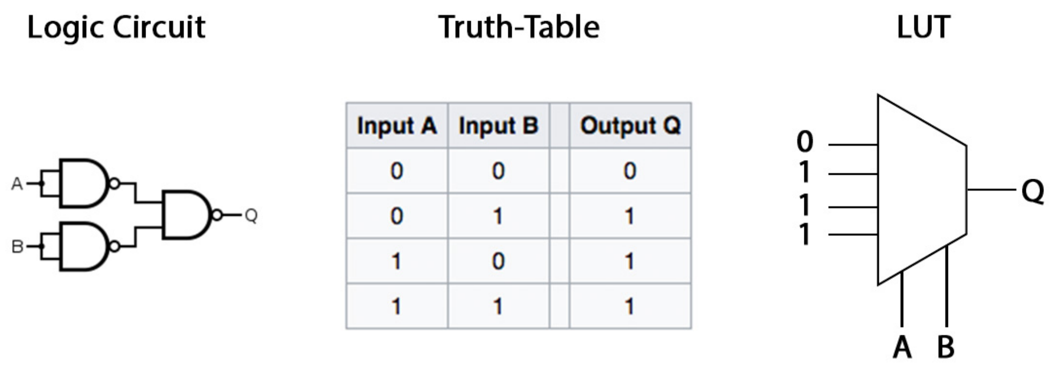
Figure 4.2: Relationship between CLBs and Slices in the Xilinx 7-Series FPGAs



Source: (XILINX, 2016)

Logic functions, and their respective truth-tables, described in the HDL code are implemented in the LUTs as multiplexers with 2 n inputs and n selectors (Fig 4.3 serves as an example).

Figure 4.3: Example of a 2-input function implemented in the LUT

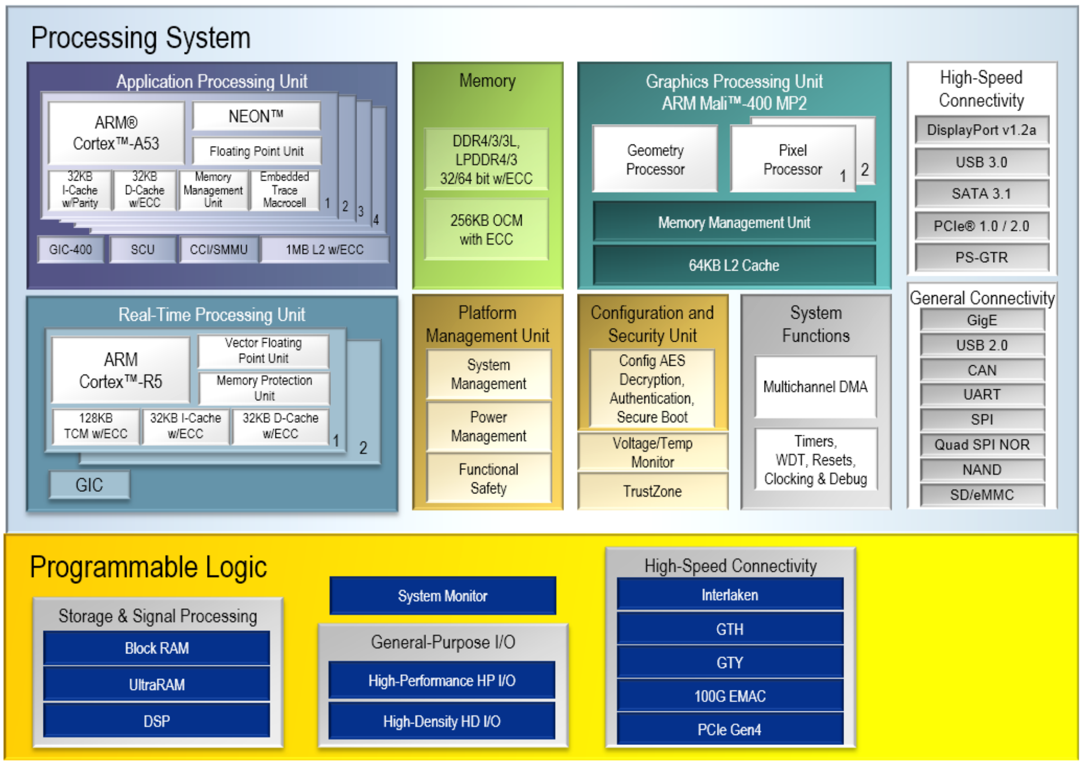


Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

4.1.2 Zynq UltraScale+

The Zynq Ultrascale+ is also a SoC device designed by Xilinx, composed of a PS and a PL part, but it has several differences compared to the Zynq-7000. Such differences start on the technology (16nm FinFET), and are really evident in terms of resources (and consequently, performance). In specific, this work utilizes the ZU9EG part, which has a quad-core A53 ARM processor and roughly 5 times as many LUTs as the entry-level Zynq-7000. The complete block diagram can be seen in Fig. 4.4

Figure 4.4: Zynq Ultrascale+'s block diagram



Source: (XILINX, 2018b)

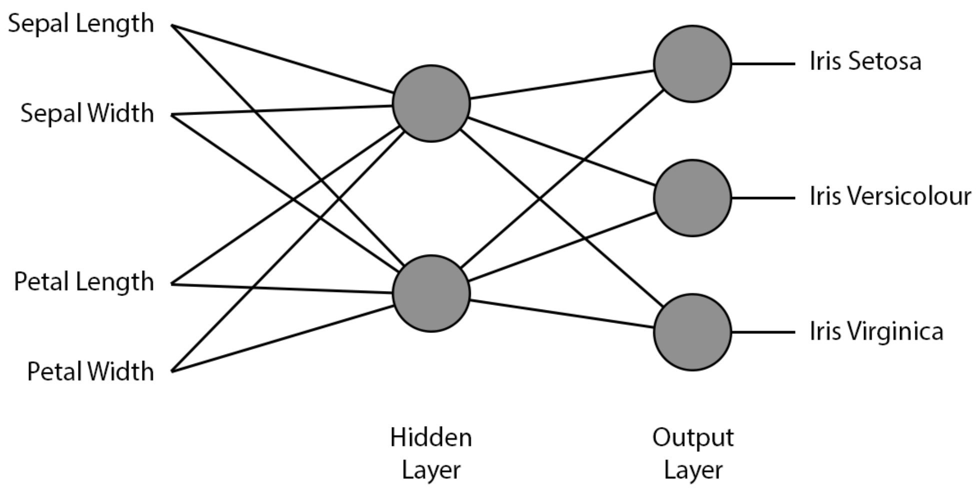
4.2 Datasets & Neural Networks

To evaluate the radiation effects on ANNs implemented on FPGAs we have chosen two networks, the Iris Flower and the MNIST. The former is a fully connected network that take features as input while the latter is a CNN that includes feature-extraction layers upstream of the fully connected layers.

4.2.1 Iris Flower

In order to evaluate fully connected ANNs, we chose the Iris Flower dataset, which is well-know in the Pattern Recognition literature, and was first introduced by (FISHER, 1936), but the actual data was collected by (ANDERSON, 1935) who wanted to quantify the morphologic variation of three related species of the Iris Flower: Setosa, Versicolor and Virginica. The dataset has a total of 150 instances, divided in three classes of 50, where each class refers to a species. The ANN gets four attributes of a given flower at the input, then, after processing is done, the actual classification is obtained by verifying which output is higher. Fig. 4.5 illustrates this ANN's very simple topology.

Figure 4.5: Iris Flower ANN's topology



Finally, Table 4.1 presents the Zynq-7000's FPGA resource utilization by our implemented designs of the ANN. Be aware that the last two lines of Table 4.1 will be explained in Chapter 7.

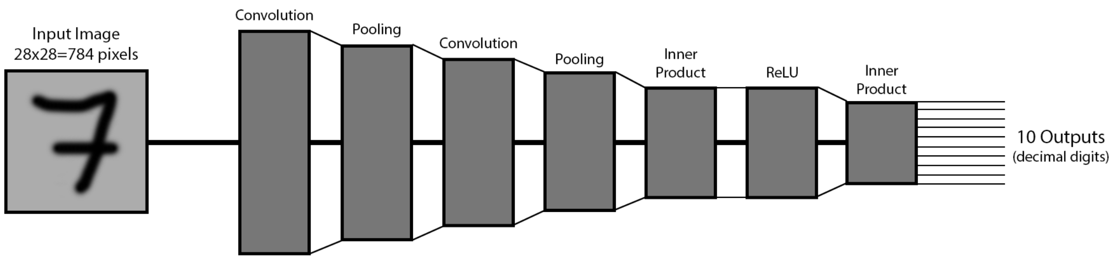
Table 4.1: Resource utilizations for the Iris Flower ANN designs in the Zynq-7000

Hardening	FF	DSP	LUT
Unhardened	3,5k (4%)	56 (25%)	16k (30%)
Selective TMR	3,7k (4%)	56 (25%)	25k (47%)
Full TMR	4,1k (4%)	168 (75%)	43k (81%)

4.2.2 MNIST

In order to evaluate neural networks with feature extraction characteristics, such as the CNNs, we chose the Modified National Institute of Standards and Technology (MNIST) dataset, which has 60,000 instances of 28x28 grayscale pixel images of handwritten digits (ranging from 0 to 9) (DENG, 2012). We have designed our CNN using a very similar topology to the popular LeNet (LECUN et al., 1998). The topology of our CNN can be seen in Fig. 4.6, and we've been able to achieve an accuracy of approximately 94% with it. It is important to highlight that we have split the dataset in 50,000 images for training and 10,000 images for testing.

Figure 4.6: MNIST CNN's topology



As we can also perceive from fig 4.6, CNNs have several types of layers, that perform different types of arithmetic operations. *Convolution* layers apply filters to the input image, through multiply-accumulate operations. *Pooling* layers focus on reducing dimensionality, and they do so by computing either the maximum or the average value within clusters of pixels. *Inner Product* layers are fully-connected, meaning that, for every output, a weighted sum of the inputs is performed. Finally, activation layers, such as *ReLU*, decide whether or not the computed value of a neuron in the previous layer is relevant and should be passed forward.

We have implemented our CNN using the parser tool described in Chapter 3. For the Zynq Ultrascale+ device, we have generated HDL code directly, prioritizing performance in exchange for high resource utilization. This design is a pure pipeline of data, with no FSM to control it. As mentioned earlier, generating HDL code directly is harder than using HLS and does not scale for larger CNNs. Nevertheless, we have implemented MNIST CNN directly in HDL in order to better evaluate, through fault injection campaigns, the sensitivity/criticality of the different layers in the neural network. Table 4.2 shows the resource utilization to implement the MNIST CNN with no FSM in the Zynq Ultrascale+ device. The last two lines of Table 4.2 will be extensively explained in Chapter 7, but as you can already see, full TMR is unfeasible resource-wise in this case, so we have completed the table with theoretical values.

Table 4.2: Resource utilizations for the MNIST CNN designs in the Zynq Ultrascale+

Hardening	FF	DSP	LUT
Unhardened	21k (4%)	0 (0%)	193k (70%)
Selective TMR	21k (4%)	0 (0%)	206k (75%)
Full TMR (Theoretical)	63k (12%)	0 (0%)	579k (210%)

Alternatively, we have also used the parser tool described in Chapter 3 to generate C code. Then, through HLS with no optimization parameters, we have generated relatively small designs of the exact same neural network. The downside in this case is that these small designs (in terms of resource utilization) take thousands of clock cycles to process a given input image. Table 4.3 shows the resource utilization by the multi-cycle versions of the MNIST CNN in the Zynq-7000 device. All of them will be referenced/explained in Chapters 6 and 7.

Table 4.3: Resource utilizations for the MNIST CNN designs in the Zynq-7000

	FF	DSP	LUT
Half Precision	12 (5%)	2062 (2%)	3270 (6%)
Single Precision	17 (7%)	2802 (2%)	4437 (8%)
Double Precision	42 (19%)	5407 (5%)	9833 (18%)
MxM Convolution	17 (7%)	3419 (3%)	6756 (12%)
ABFT MxM Convolution	17 (7%)	5355 (5%)	11259 (21%)
SmartPool	17 (7%)	3184 (3%)	5503 (10%)

5 EXPERIMENTAL METHODOLOGY

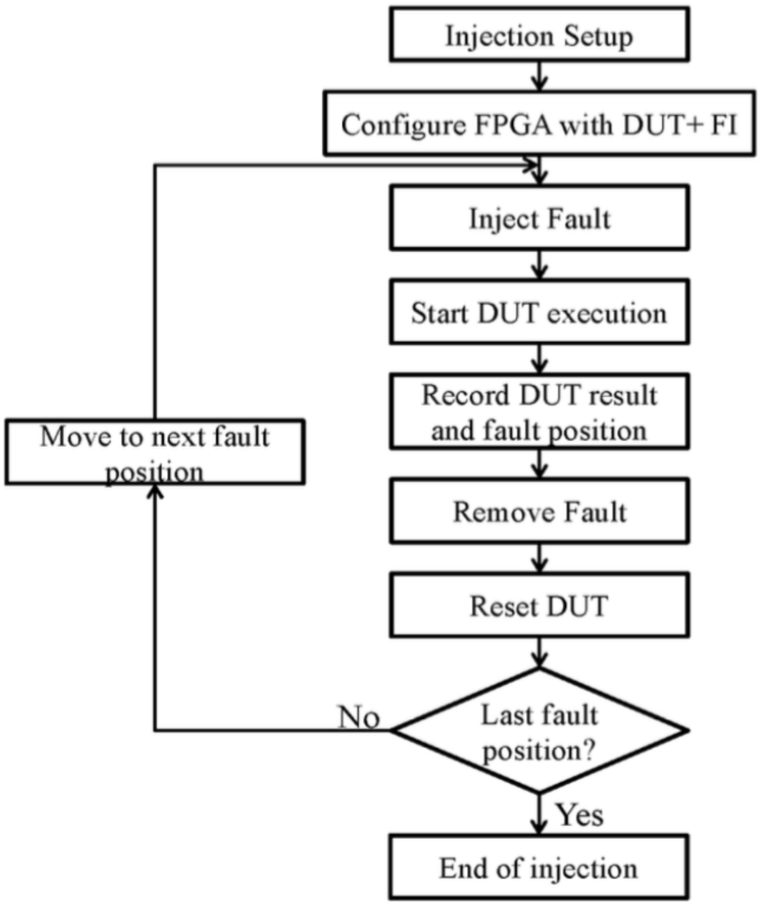
Among the methods to evaluate SEEs on electronic devices, the most realistic one is to take the device to its real application environment (satellites in space, for example). Sometimes however, due to practical and/or financial constraints, this approach is not an option. Moreover, it may take large periods of time, or a very high number of devices in parallel, to collect a statistically relevant amount of data. As a result, accelerated radiation tests are the most common way to qualify the reliability of integrated circuits for SEEs (JEDEC, 2006). In these experiments the devices are exposed to a radiation source with much higher density than the normal levels it would experience in its application environment, reducing drastically the amount of time necessary to obtain a statistically significant amount of useful data. Accelerator facilities provide a variety of particles, such as neutrons, protons, and heavy ions. Neutron facilities like the Los Alamos National Science Center (LANSCE) in the United States, and the Rutherford Appleton Laboratory (RAL) in the United Kingdom are generally used for testing devices to be used in terrestrial and avionic applications. Proton facilities as the Paul Scherrer Institut (PSI) in Switzerland are capable of generating protons with enough energy to simulate solar flares. Heavy ions facilities like the 8UD Pelletron at Universidade de São Paulo (USP), are usually for evaluating devices destined to space and deep space. Finally, there are facilities in which the Device Under Test (DUT) is exposed to a very high flux of a cocktail of particles. The best example of this facility is the CERN High Energy Accelerator Mixed-field (CHARM) located in Switzerland.

A third method of qualification is fault injection by emulation or simulation. This method is, at the same time, the less costly and the most flexible, but it does not waive the necessity of a radiation test. Usually, fault emulation is an attractive technique to predict the susceptibility of a system before submitting the device to an accelerated test, for example. This approach basically consists in flipping bits of memory components of FPGAs and processors through the assistance of an embedded circuit or a monitoring computer. Single Event Upsets (SEUs) can be emulated randomly or sequentially (when every configuration bit is flipped in a sequential order). Thus, fault injection provides deeper information when compared to radiation experiments, since the correlation between the flipped bit's location and the fault effect is known. In this work, we exploit both fault injection and radiation experiments, which are described in Sections 5.1 and 5.2, respectively, along with the main metric of each experiment.

5.1 Fault Injection

This work uses two fault-injection frameworks. For the Iris Flower ANN, we use the one presented in (TONFAT et al., 2016), which is based on the Internal Configuration Access Port (ICAP) block of Xilinx FPGAs, an ICAP controller and a monitor computer. The ICAP block can read and write the configuration memory of the FPGA using the readback and dynamic reconfiguration capabilities of the FPGA. Hence, the fault-injection framework can emulate SEUs in the configuration memory of the FPGA. The flow of a fault-injection campaign is presented in Fig. 5.1. The first task is the definition of the injection area and the type of fault-injection campaign. For the case considered in this work, the injection area is the area of a given neuron of the ANN, implemented in the FPGA and an exhaustive fault-injection campaign is selected to known which configuration bits produce errors on the outputs of the artificial neural network. The exhaustive fault-injection campaign produces bit-flips on all the configuration bits of the injection area, one at a time.

Figure 5.1: The exhaustive fault-injection flow



For the MNIST CNN implemented on the Zynq-Ultrascale+ device, we use the Processor Configuration Access Port (PCAP) of the MPSoC system and emulate SEUs by inserting upsets through software executed on the A53 quad-core processor. We isolate each of the neural network layers into separate regions of the device using *Pblocks*. The location of the injected fault can be correlated to the Pblock and thus a specific layer of the network. Our fault-injection campaign randomly injected about 13 million faults, but it is important to highlight that the area is comprised of only the configuration bits related to CLBs (LUTs, DSPs, FFs and interconnections).

The fault injection test, as in any higher abstraction level simulation, cannot measure the cross section, since it depends on the physical-level sensitivity to radiation, as defined in Chapter 6.1. Fault injection aims to measure the Architectural Vulnerability Factor (AVF), which is the probability for an injected fault to propagate through the system and manifest at the output as a visible error (MUKHERJEE et al., 2003). It is simply obtained by dividing the number of observed errors (N_{Errors}) by the number of injected faults ($N_{Injections}$):

$$AVF = \frac{N_{Errors}}{N_{Injections}}$$

5.2 Radiation Experiment

Radiation experiments were conducted with neutrons at the Los Alamos Neutron Science Center (LANSCE) facility of the Los Alamos National Laboratory (LANL) and the ChipIR facility at ISIS, Didcot, UK. Both LANSCE and ChipIR provide a neutron spectrum that mimics the atmospheric neutron one and have already been demonstrated to be suitable to emulate terrestrial neutrons effects in electronic devices and systems. To avoid any result bias due to the facility we tested the Zynq-7000 implementing the Iris Flower ANN only at LANSCE. At ChipIR, we've tested the pure pipelined version of the MNIST CNN on the Zynq Ultrascale+, and all the multicycle versions of the same neural network (mentioned in 4), in the Zynq-7000.

The neutron flux at LANSCE and ChipIR is about 8 to 9 orders of magnitude higher than the terrestrial flux $(13n/(cm^2.h))$ at sea level (JEDEC, 2006) (CAZZANIGA; FROST, 2018). The Zynq-7000 device was tested for approximately 40 hours at LANSCE, for a total fluence of $1.6x10^{11} neutrons/cm^2$, which is equivalent to about 1.5 million years of natural exposure. The Zynq Ultrascale+ was tested for about 20 hours at ChipIR, for a total fluence of $3.0x10^{11} neutrons/cm^2$, which is equivalent to more than 2.5 million years of natural exposure. Finally, he Zynq-7000 was tested for about 30 hours at ChipIR, for a total fluence of $4.0x10^{11} neutrons/cm^2$, which is equivalent to more than 3.5 million years of natural exposure. Our experimental setups on the beam lines at LANSCE and ChipIR can be seen figures 5.2 and 5.3, respectively.

Figure 5.2: Radiation experiment at the LANSCE facility of the Los Alamos National Laboratory, USA.

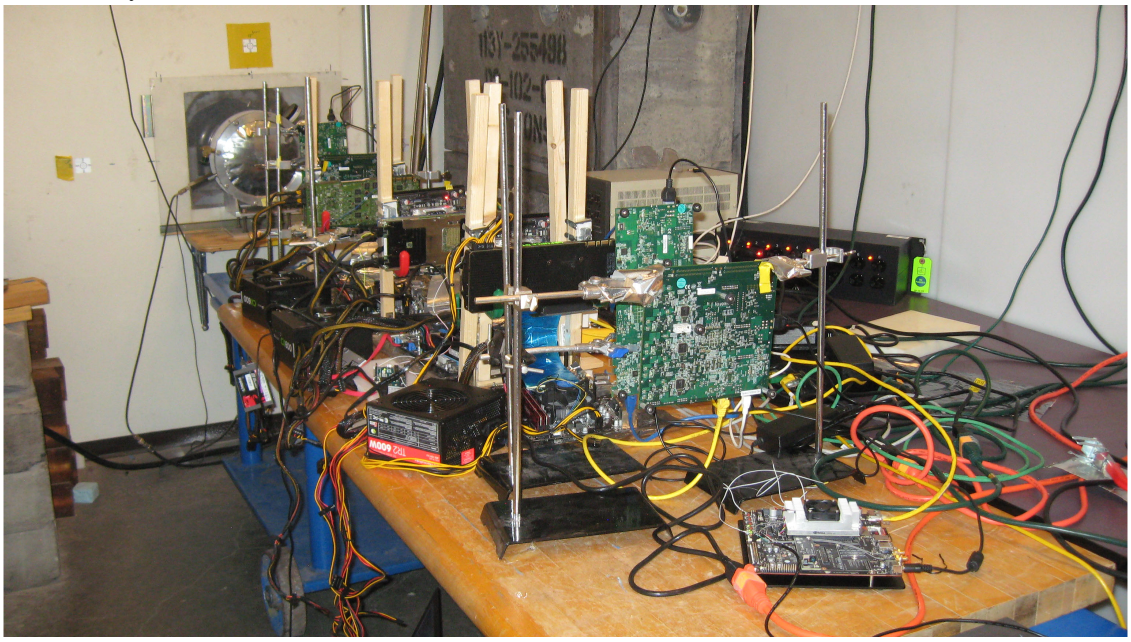
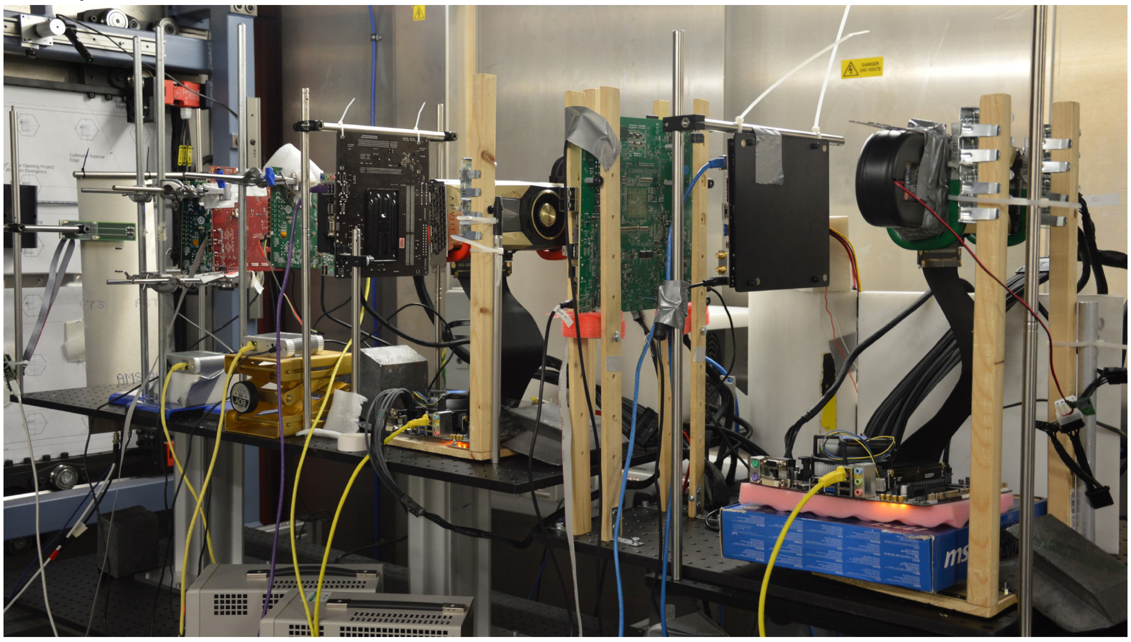
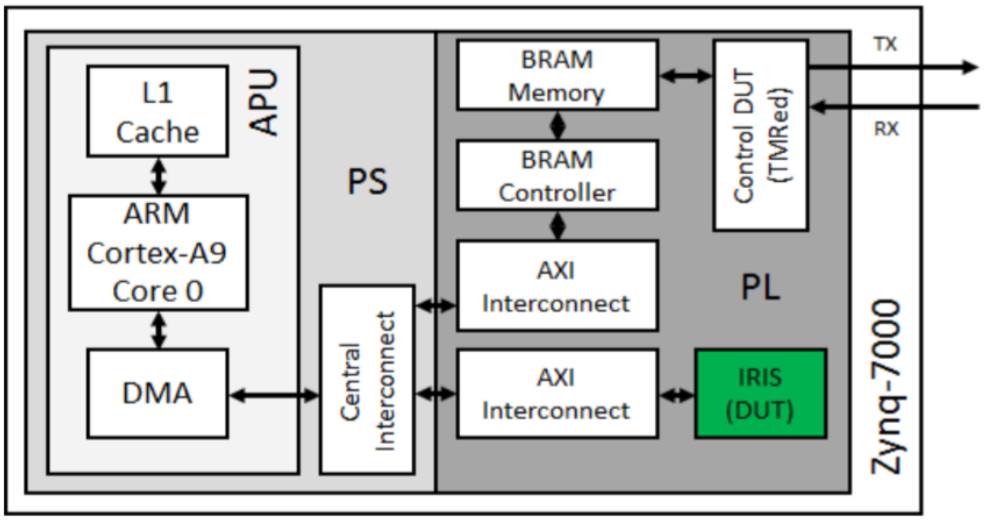


Figure 5.3: Radiation experiment at the ChipIR facility of the Rutherford Appleton Laboratory, UK.



A heterogeneous setup based on the PS and PL parts of the Zynq-7000 was developed for testing the Design Under Test (DUT), as illustrated in Fig. 5.4. On the PS part, a software running on one of the ARM Cortex-A9 cores controls the DUT. It is worth highlighting that the L2 Cache of the PS part was disabled to not compromise the processor's reliability, while the L1 Cache of the processor was left enabled to not compromise the processor's performance. Such choice was based on the results obtained by (TAMBARA et al., 2016). The communication between PS and PL parts is performed through General Purpose (GP) ports, which provide a very high data throughput. On the PL part, the Control DUT hardware block monitors the memory space of the DUT. This block is implemented with TMR aiming to mask SEUs. If the block detects errors, it sends them to a script running on a monitor computer, which time-stamps the errors and logs them for future analysis. For the Zynq Ultrascale+ we adopt a very similar setup, where the PS is also responsible for controlling the DUT, but the difference here is that the logs are transmitted via Ethernet connection to the monitoring computer.

Figure 5.4: Block diagram of the developed setup for testing the DUT



In a radiation experiment, the most important metric is the Cross Section, which quantifies the sensitivity of a system to particles (JEDEC, 2006). The cross section is measured dividing the number of observed errors (N_{Errors}) by the particles fluence (average flux multiplied by the total time of the experiment).

$$CrossSection = \frac{N_{Errors}}{Fluence}$$

5.3 Complimentary Nature of Experiments

When performing radiation experiments, the entire device is irradiated and particles affect the device with realistic error rates, providing a more accurate prediction of the error rate. However, errors are observed only when they appear at the output, making it impossible to deeply evaluate fault propagation in the tested design or device. On the other hand, when injecting faults, we corrupt specific portions of the hardware, in order to correlate the fault source to the observed effect in the output. Beyond that, the cross section is measured with the unit of area (cm^2), while the AVF, being a probability, is dimensionless. This means that radiation and fault injection provide complementary information about the reliability of the system being tested. Thus, we can say that one experiment does not exempts the execution of the other.

To provide a deep and complete understanding of the reliability behaviors of neural networks in FPGAs, we've performed both beam experiments and fault injection, adopting an iterative approach. First, we do preliminary beam experiments to evaluate the sensitivity of the designs and to analyze the kind of errors that radiation induces on FPGAs running ANNs. Then, we use fault injection to find out which specific parts of our design are the most vulnerable to radiation. With this information at hand, we develop and apply efficient hardening strategies. Finally, we go through another round of beam experiments to validate our mitigation proposals.

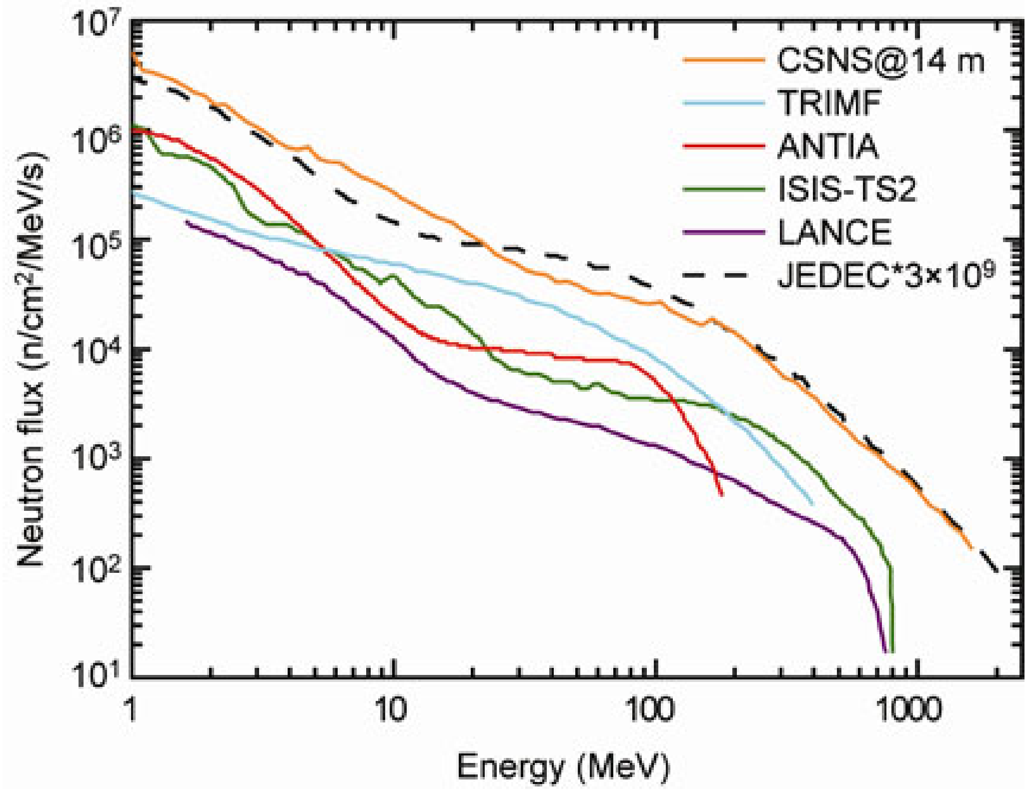
6 RELIABILITY EVALUATION

In this chapter we present our preliminary radiation results and our fault injection data. We start with the report of our beam experiments to get a general sense of the error rate and criticality. Then we analyze vulnerability aspects of neural networks by showcasing our insights from fault injection. Statistical error, at 95% confidence, is lower than 15% for all reported values.

6.1 Radiation Experiment

Before presenting experimental data, it is important to highlight that, on both test facilities (LANSCE and ChipIR), the neutron flux on the beam lines have very similar energy spectrum patterns to the terrestrial neutron flux. On Fig. 6.1, we can see the energy spectrum of neutron fluxes from different testing facilities around the world. Note that ChipIR is mentioned as *ISIS-TS2* in 6.1. As such, by testing the devices in these facilities we can get an accurate estimate of their error rates in realistic applications, simply by multiplying the obtained cross sections by the average flux at which the device is exposed (usually around $13n/(cm^2 \times h)$).

Figure 6.1: Neutron flux energy spectrum, considering multiple testing facilites

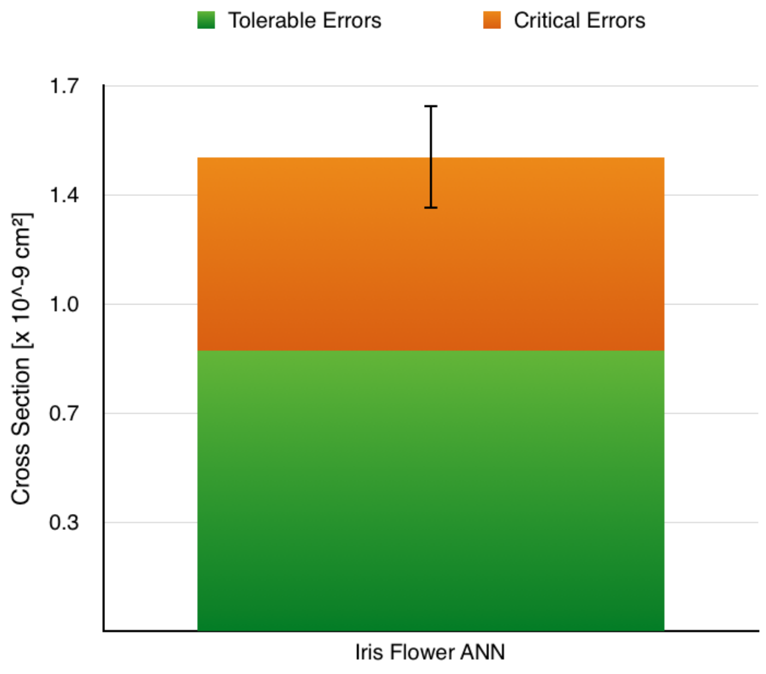


6.1.1 Iris Flower

The Iris Flower ANN implemented on the Zynq-7000 was tested at LANSCE in October 2017. We tested the device for 40 hours, accumulating a total fluence of more than $1.6x10^{11} neutrons/cm^2$ and gathering more than 400 events. Beam results with the Iris Flower ANN are shown in Figure 6.2 in terms of cross section (i.e. the probability for an impinging neutron to generate an observable error). Reported data shows a predominance of tolerable errors over critical errors (to be precise, 59% of all errors were considered tolerable). This promising result for neural networks reliability confirms our argument about the approximated computing aspect of ANNs from Chapter 2.6, where we said that not every output error is significant enough to jeopardize the correctness of the application.

If we multiply the cross section by the average terrestrial flux (13 $n/(cm^2 \times h)$), we end up with an expected error rate of about $1.68x10^{-3}$ errors per year. While this error rate may seem reasonable, it is worth mentioning that it equals to about 20 FIT (Failure In Time, i.e., errors every 10^9 hours of operation). The ISO 26262 for safety-critical features as object detection imposes the FIT rate to be lower than 10 (without making a clear distinction between tolerable and critical errors). So, even a very simple design as the Iris Flower may then not be sufficiently reliable to be used in automotive applications. In Chapter 7 we will present and discuss hardening methods that can significantly reduce the cross section, consequently reducing the error rate of the Iris Flower ANN as well.

Figure 6.2: Neutrons cross section of the Iris Flower ANN



6.1.2 MNIST

The MNIST CNN implemented on the Zynq-7000 was tested at ChipIR in June 2018. We tested the device for 30 hours, accumulating a total fluence of more than $4.0x10^{11} neutrons/cm^2$ and gathering more than 1,000 events. The main objectives for MNIST were to evaluate the radiation effects on feature extraction, the data precision impact on the network reliability, and to analyze the different behavior of convolution when implemented in the standard way or as matrix multiply.

6.1.2.1 Data Precision

As stated on Chapter 3.2, using C code and HLS, we can easily change the datatype in our designs, which enables us to explore reduced-precision configurations for improved reliability (but always keeping in mind that the accuracy of the trained neural network shouldn't be significantly compromised). So, we have decided to test three levels of precision with floating point: double, single, and half precision (64, 32, and 16 bits, respectively). With our MNIST, both double and single precision floating point versions of the neural network achieved the same accuracy (94%), meaning that using 64 and 32 bits, we never had an overflow while executing the arithmetic operations within the layers. However, when we reduced to half precision, the accuracy of our model dropped by 2%.

In terms of reliability, we can see that the cross section trend in Fig. 6.4 is very similar to the resource utilization graph (Fig. 6.3), which was expected. In fact, when reducing precision we take good care in not changing anything in terms of arithmetic operations. Thus, the only thing that changes is the area of the design and not the circuit computing nature. The lower the exposed area, the lower the cross section. Once again, critical errors were the minority across all levels of data precision. Moreover, the cross section of critical errors increased 11% when going from 64 to 32 bits, and then increased an additional 4% when going from 32 to 16. This corroborates our hypothesis in Chapter 3.2 that, as we reduce precision, we are more likely to see significant output errors (due to the increase architectural vulnerability).

With that said, the insight from our results would be to reduce precision as much as possible, as long as the accuracy doesn't drop significantly in the process, and as long as the percentage of critical errors doesn't rise significantly in the process.

Figure 6.3: Resource utilization by the MNIST CNN using three different levels of data precision

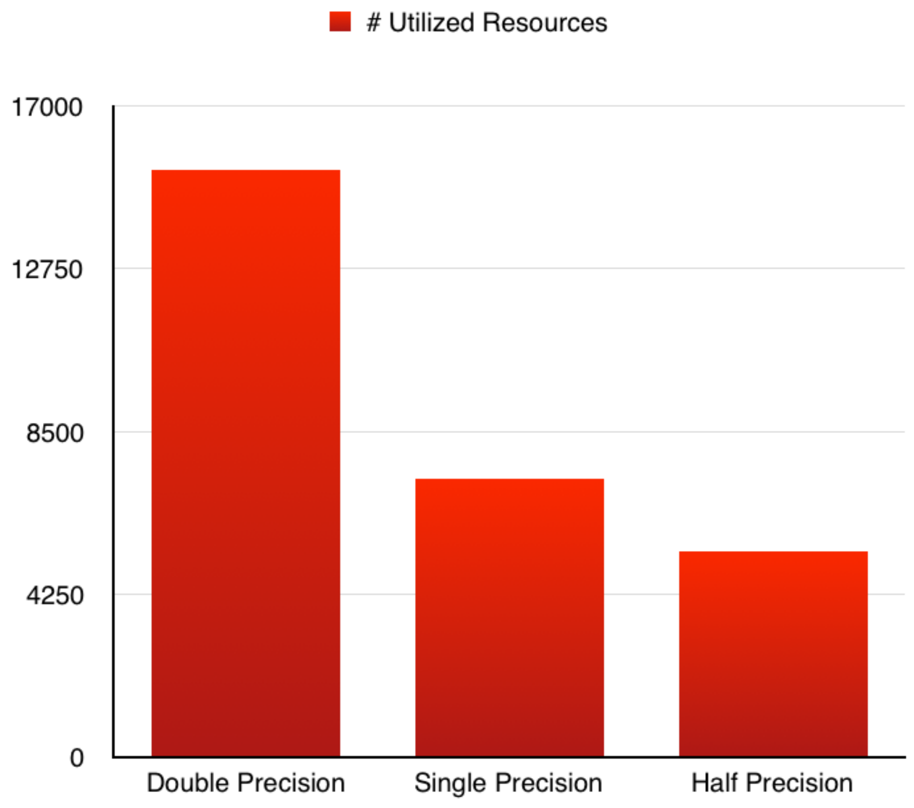
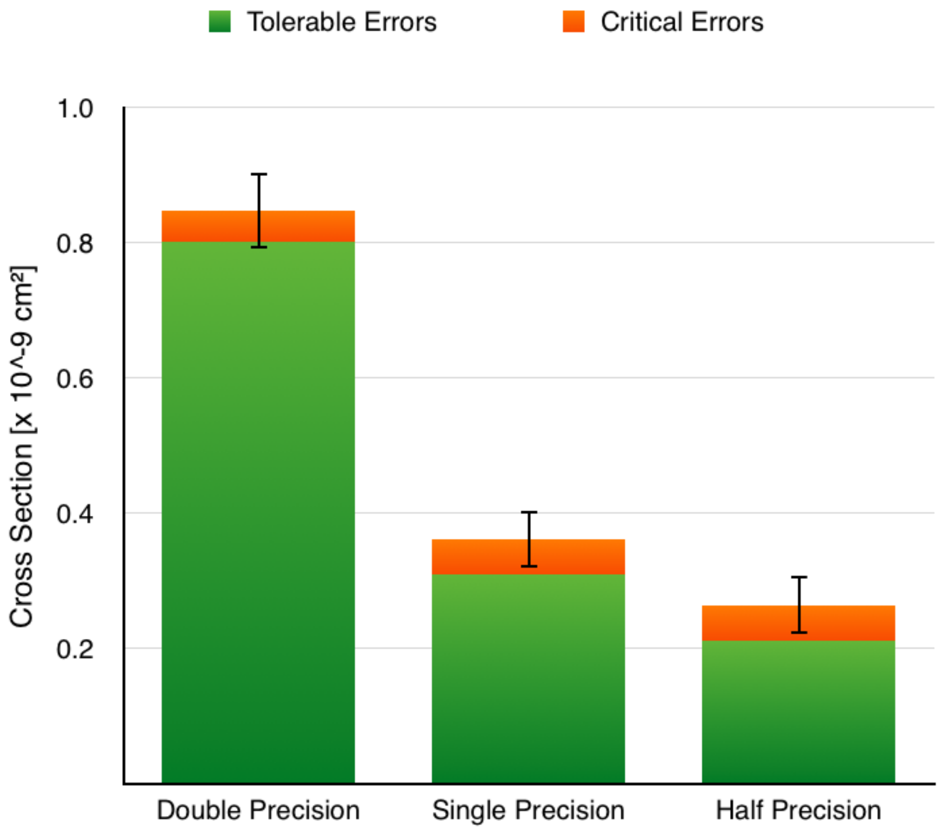


Figure 6.4: Neutrons cross section of the MNIST CNN using three different levels of data precision



6.1.2.2 Regular Convolution vs. MxM Convolution

As introduced in Chapter 2.3, another aspect of CNNs we wanted to evaluate is the impact of modeling convolutional layers as matrix multiplications (VASUDEVAN; ANDERSON; GREGG, 2017), as this is a very effective strategy in GPUs and parallel processors (RIZVI; CABODI; FRANCINI, 2017). Dissimilarly to what we have observed with data precision, however, the cross section trend does not follow the resource utilization one, as we can see in Figures 6.6 and 6.5. In fact, while resource utilization increases by about 40%, the total cross section decreases by 3%. This is justified by that fact that we have changed the algorithm that implements the convolutional layers, and consequently the set of arithmetic operations that produces the CNNs output. It is interesting to notice that, if we look exclusively to the critical errors, the cross section increases by almost 30% when convolution is implemented with matrix multiplication. This suggests that the circuit implementing matrix multiply is more likely, once corrupted, to generate errors that significantly impact the neural network behavior.

If we looked exclusively to the total cross section, we would conclude that using matrix multiplications is a better choice in terms of reliability. However, if we looked exclusively to the rate of critical errors, we would conclude otherwise. In order to get a view of the bigger picture here, we should point out that, in addition to using more resources, the MxM version also takes more clock cycles to process one input image, as Fig. 6.7 illustrates. Thus, as an attempt to determine which design actually offers the best reliability, we've calculated the Mean Executions Between Failures (MEBF) metric (RECH et al., 2014), which says, on average, how many successful executions can be completed between errors (so, the higher the better). As we can see in Fig. 6.8, regular convolution comes out on top (even though it is not by a large margin).

Figure 6.5: Resource utilization by the MNIST CNN using two different types of convolutional layers

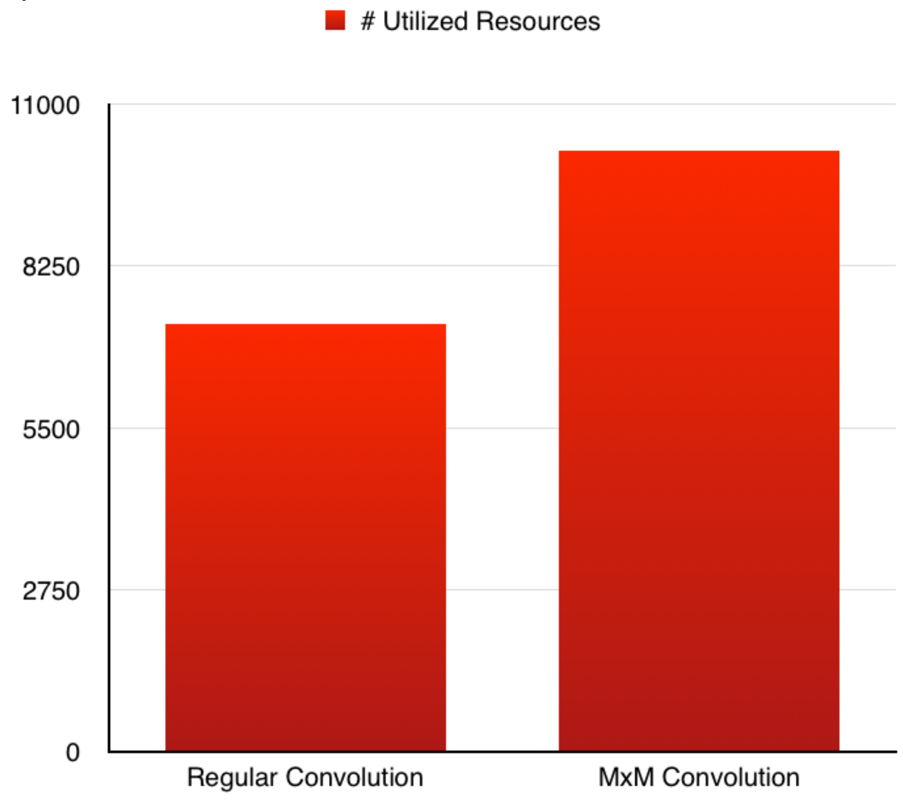


Figure 6.6: Neutrons cross section of the MNIST CNN using two different types of convolutional layers

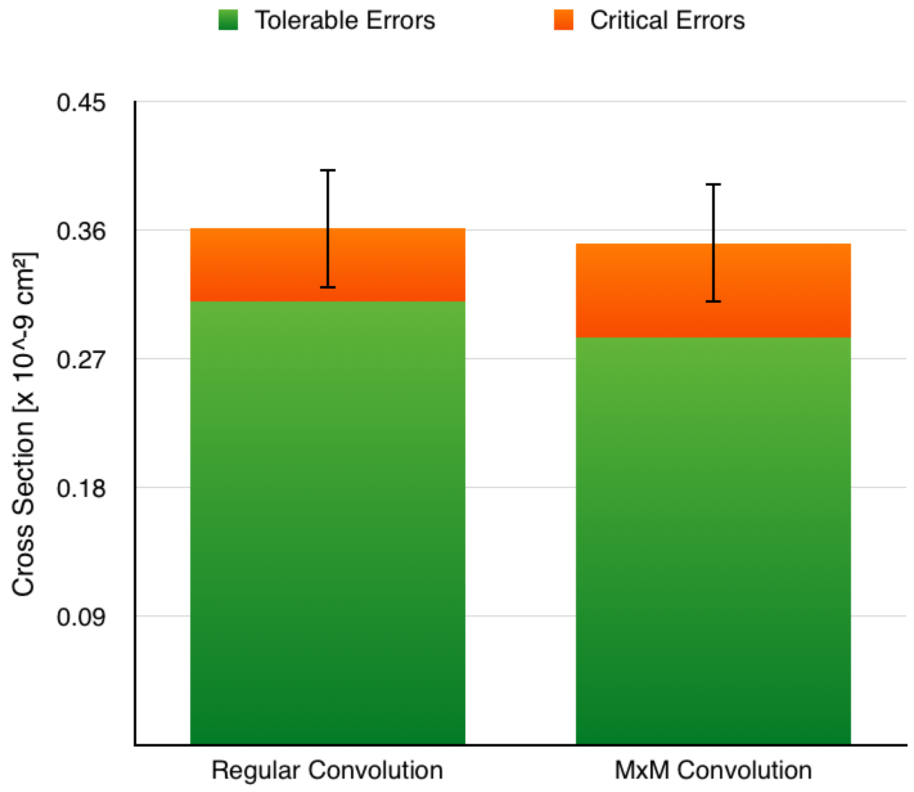


Figure 6.7: Clock cycles taken by the MNIST CNN using two different types of convolutional layers to process one input image

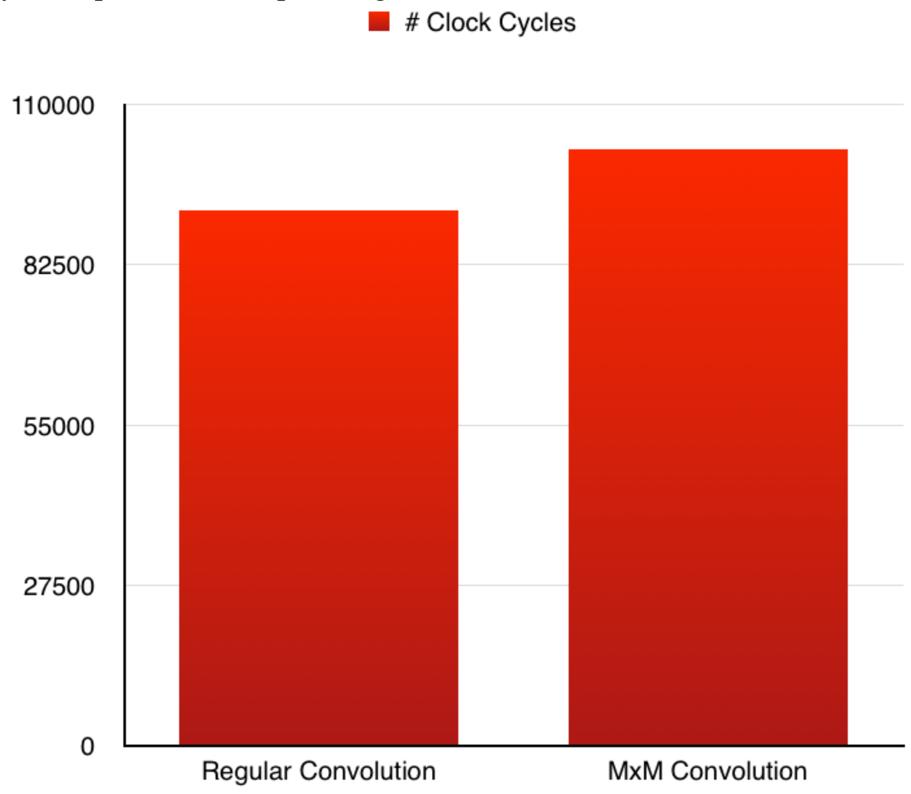
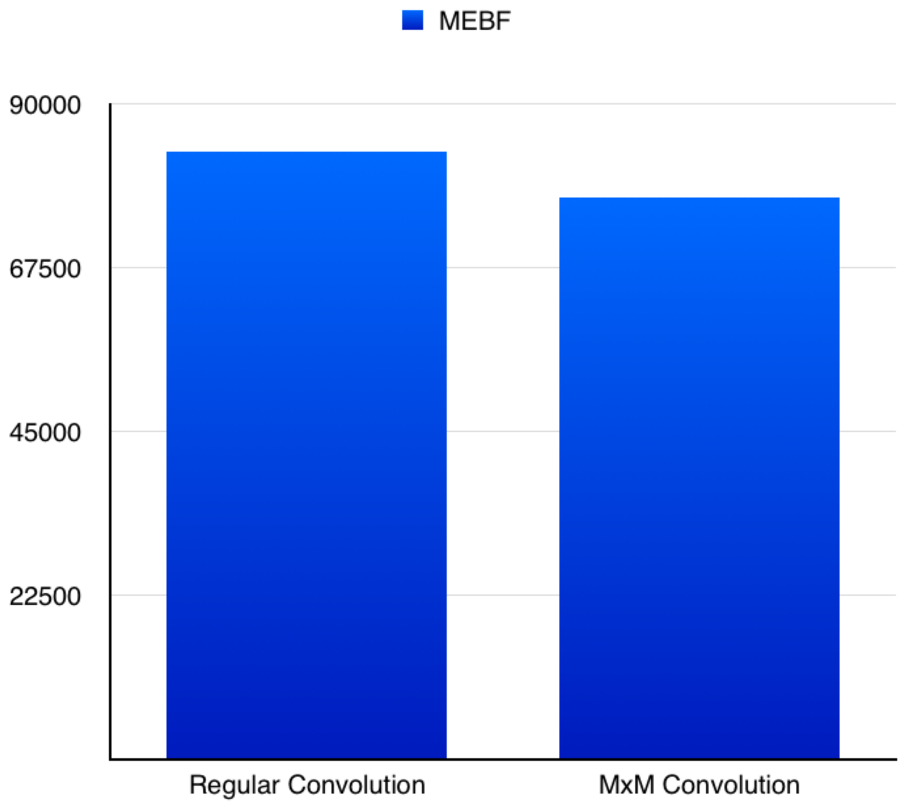


Figure 6.8: MEBF of the MNIST CNN using two different types of convolutional layers



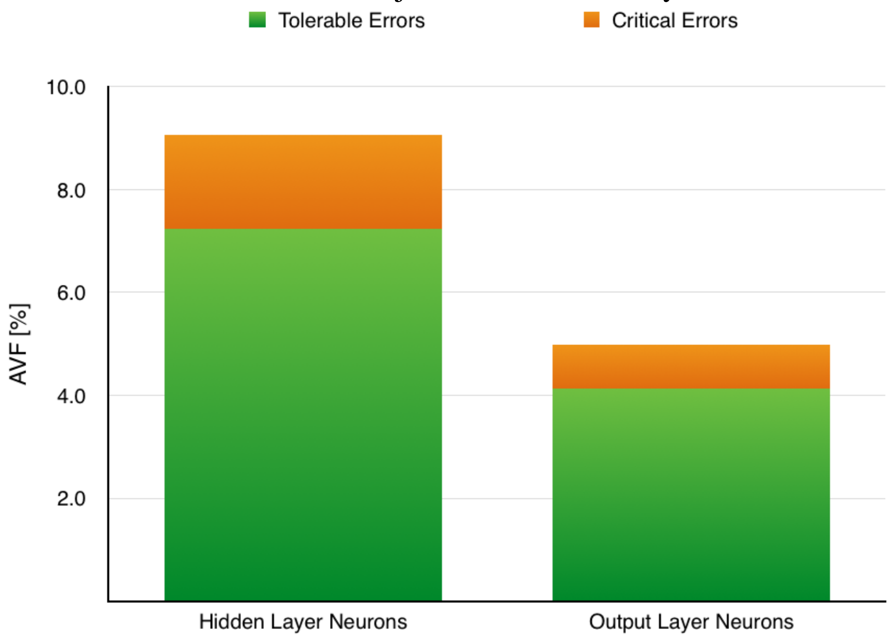
6.2 Fault Injection

While offering several advantages, radiation experiments allow little visibility of error propagation. As radiation-induced errors are detected only at the output, it is not possible to correlate the observed errors with the source of corruption. We take advantage of the fault-injection frameworks described in Chapter 5.1 to better understand the observed phenomena. We decide to inject faults separately in different layers of the networks. Our intent is to comprehend the impact of faults in layers and their propagation throughout the neural networks. It is worth noting that a direct comparison between fault-injection and radiation results is not possible. In fact, while beam experiments provide the probability of errors to occur, fault-injection evaluates the probability of a fault to affect the output, providing deeper insights on the vulnerability of ANNs.

6.2.1 Iris Flower

Fig. 6.9 shows the results obtained through fault injection in the Iris Flower ANN, expressed as AVF, i.e., the percentage of injections that propagate to the output. We divide the fault-injection outcomes into tolerable and critical errors. As discussed in Chapter 2.6, only the errors that affect the output significantly enough to alter the classification are considered critical (consequently, all others are marked as tolerable). Faults were separately injected in the Hidden Layer (HL), and Output Layer (OL) of the ANN.

Figure 6.9: AVF calculated from fault injection in different layers of the Iris Flower ANN



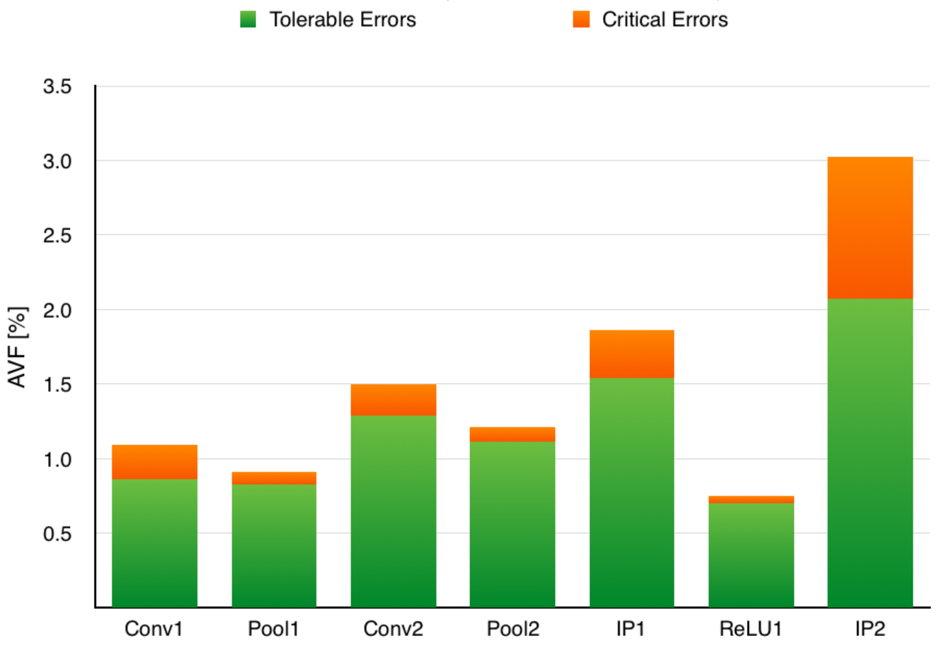
From Fig. 6.9, it is clear that, as we saw in the beam experiments, critical errors are the minority. In fact, tolerable errors were about four times more frequent than critical errors. In addition to that, we see that faults in the hidden layer are more likely to generate tolerable and critical errors in the output. This is because when a fault is closer to the input, it has more room to propagate through the fully-connected topology of the ANN. This result is promising because it gives us a starting point on what we should prioritize when implementing hardening strategies. We will further discuss this in Chapter 7.

6.2.2 MNIST

In Fig. 6.10 we present the AVF for MNIST CNN layers. Unlike the Iris Flower ANN, the MNIST CNN is composed of layers with different functions, and sizes. Thus, the sensitivity of each layer depends on position within the network, type and size.

As shown in Fig. 6, even for MNIST, the dominance of tolerable errors over critical errors holds, suggesting that this result could be a general characteristic of pattern recognition feed-forward neural networks. Additionally, it is evident that a fault in *InnerProduct2* layer (IP2) has a considerably greater chance of propagating to the output. Another insight that might be generically true is that the full connected layers are more sensitive than the ones responsible for feature extraction (convolution and pooling). This suggests that we should prioritize the hardening of certain layers in the network, as opposed to treating the ANN as an unit. We expand on this idea in Chapter 7.

Figure 6.10: AVF calculated from fault injection in different layers of the MNIST CNN



7 HARDENING TECHNIQUES

This chapter presents and discusses our attempts to improve the reliability of neural networks in FPGAs, based on data and discussion from the previous chapter. We have tested a number of different hardening strategies: from fault tolerant layers to modular redundancy. We evaluate advantages and disadvantages for all of them. Statistical error, at 95% confidence, is lower than 15% for all reported values.

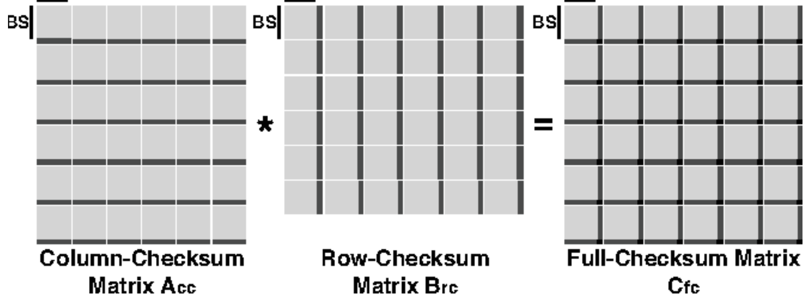
7.1 ABFT Layers

At first, we took advantage of our HLS tool to implement ABFT techniques in some types of layers of the MNIST CNN. This turned out to be a reasonably easy strategy to adopt (implementation-wise) since we just had to adapt existing C code, instead of trying to incorporate the ABFT using HDL directly.

7.1.1 ABFT MxM Convolution

ABFT in matrix multiplications is one of the most well-know hardening strategies (HUANG; ABRAHAM, 1984), and has been proved to be efficient in GPUs (RECH et al., 2013). It consists on adding one extra row to one input matrix, and one extra column to the other input matrix. Each cell in the extra row contains the sum of the elements in that respective column. Likewise, each cell in the extra column contains the sum of the elements in that respective row. Then, after the multiplication operation is complete, we can use the checksums in the result matrix to detect and correct single errors. Fig. 7.1 provides an overview of this technique.

Figure 7.1: Overview of the ABFT matrix multiplication


BS
BS
BS
BS
BS
BS
*
=
Column-Checksum Matrix A_{cc}
Row-Checksum Matrix B_{rc}
Full-Checksum Matrix C_{fc}

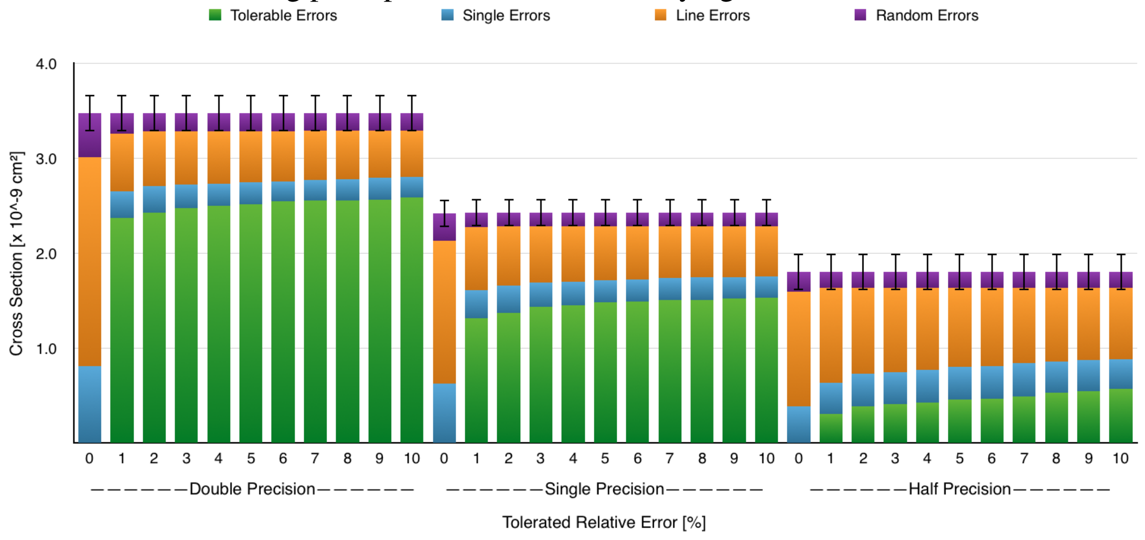
Source: (BRAUN; HALDER; WUNDERLICH, 2014)

In order to verify whether or not this ABFT technique could be effective in the convolutional layers of neural networks, we first tested a pure matrix multiplications design, using matrices of similar sizes to the ones in our MNIST CNN topology. We've also performed radiation experiments with different precisions of floating point. Here, we introduce the concept of Tolerated Relative Error (TRE), which basically specifies an interval of tolerance for how much the output can deviate from the expected value and still be considered as a tolerable error. For example, if we set a TRE of 10%, our output needs to be greater than 90% of the expected value, but no higher than 110% of the expected value, to be considered a tolerable error. If our output is outside the interval of tolerance, we then classify it as one of the following:

- Single Errors = Only one element in the result matrix is wrong
- Line Errors = One entire row/column in the result matrix is wrong
- Random Errors = All other scenarios

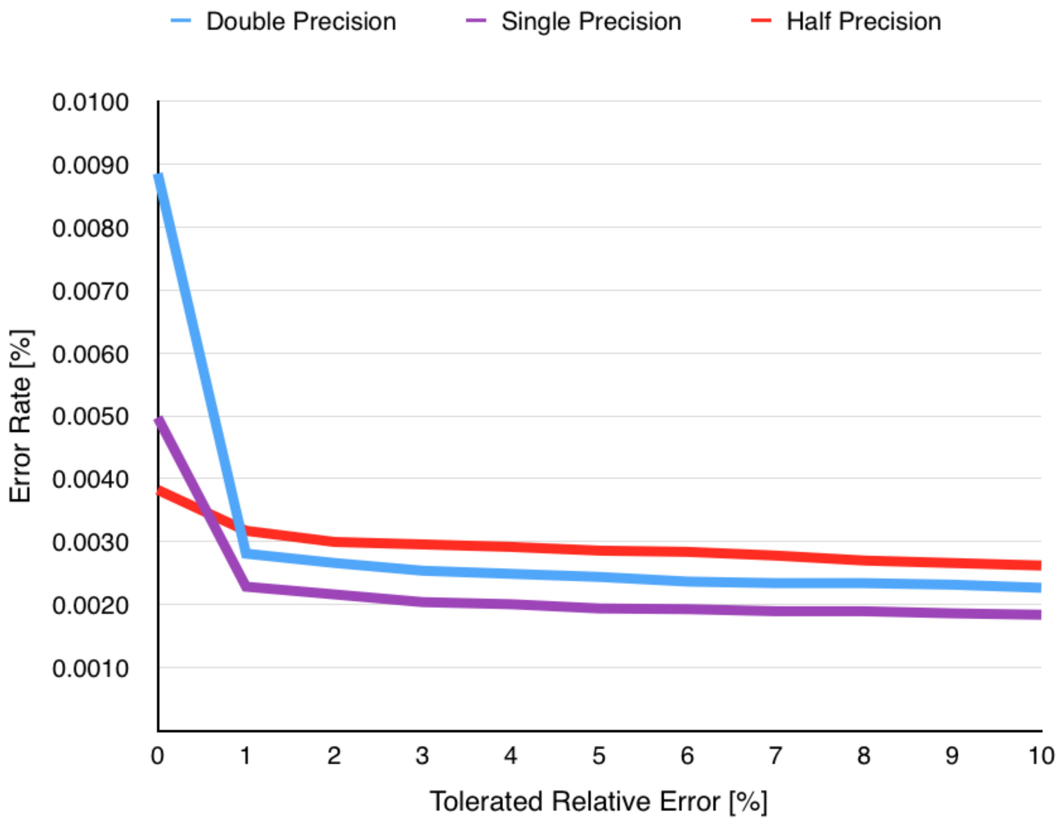
Fig. 7.2 presents the neutron cross section of the matrix multiplication designs. As previously stated, we've tested three levels of precision with floating point data, and we've varied the TRE from 0% to 10%.

Figure 7.2: Neutrons cross section of the matrix multiplications designs, with three different levels of floating point precision, and TRE varying from 0% to 10%



As expected, the cross section decreases when we reduce precision. This is because of the lower amount of resources required to implement low precision matrix multiplications, thus reducing the exposed area and the cross section. In addition to that, the fault model, meaning the frequency of occurrence for all types of errors is nearly the same for the three designs. Considering the TRE as 0%, we see that about 20% of all errors are classified as single errors, meaning that only one element in the result matrix is wrong, and that ABFT would be able to correct it. Unfortunately, most of the non-tolerable errors are line errors, which usually means that the inputs were corrupted (each element of the input matrix is used to calculate a whole column or row of the output matrix), and ABFT can't do anything about it in this case. Another interesting fact is that, increasing the TRE more than 1% doesn't significantly increase the percentage of tolerable errors, meaning that whenever an element of the output matrix is wrong, its value is extremely different from the expected value. In Fig. 7.3 we can see the error rate (i.e., percentage of erroneous executions) of the three tested designs. An interesting insight here is that, after increasing the TRE to 1%, both double and single precision designs have a lower error rate than the half precision one, despite having higher cross sections.

Figure 7.3: Error rate of the matrix multiplications designs, with three different levels of floating point precision, and TRE varying from 0% to 10%



With all this information at hand, we've proceeded to perform beam tests with ABFT applied to the convolutional layers of the MNIST CNN. Fig. 7.4 shows the resource utilization numbers, while Fig 7.5 shows the cross section. We can see that ABFT has managed to reduce the cross section by about 20%, even though it uses almost 67% more resources than the unhardened version.

Figure 7.4: Resource utilization by the MNIST CNN using matrix multiplication

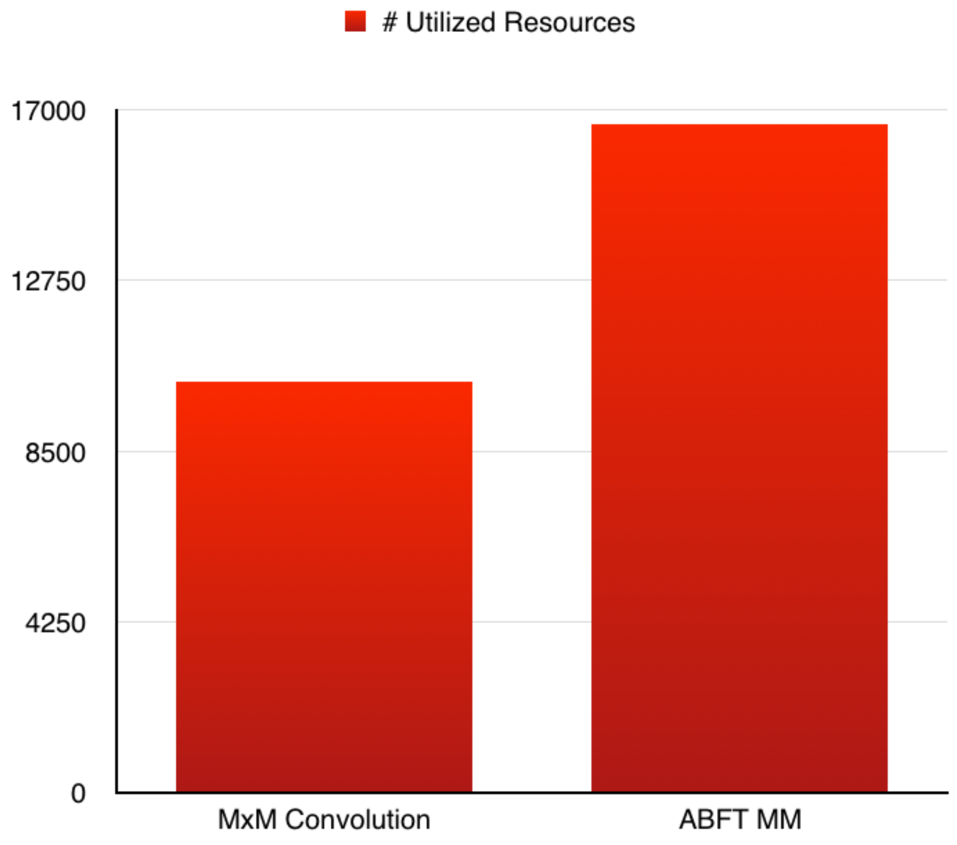
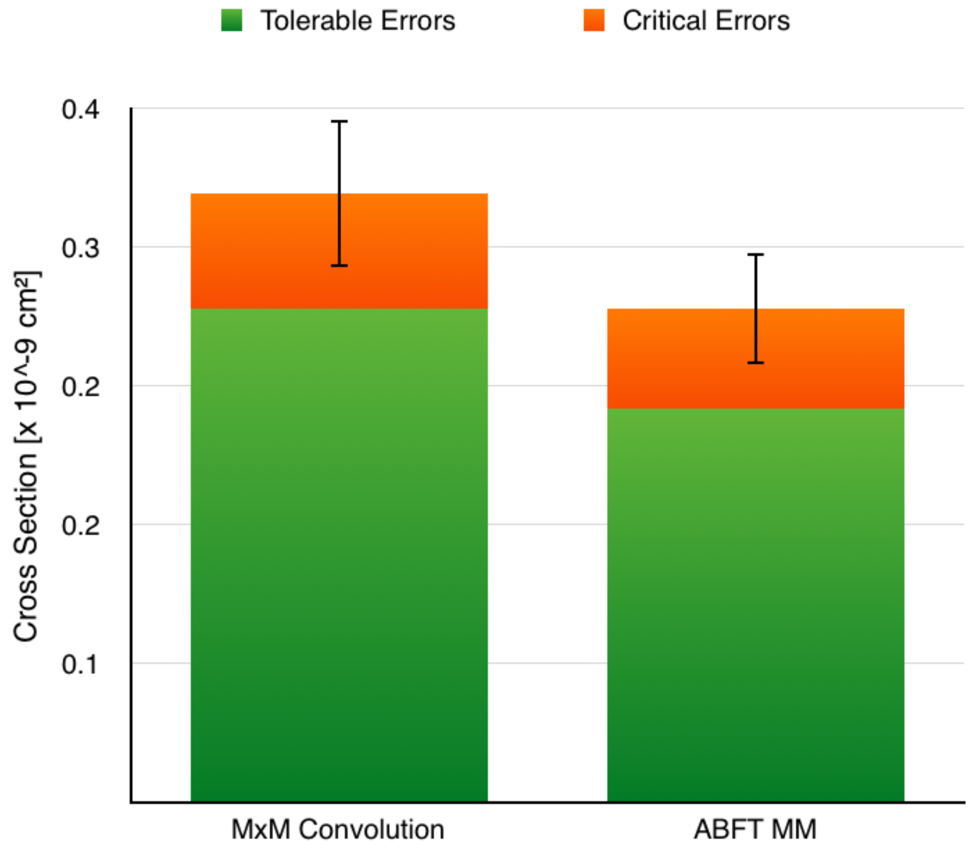


Figure 7.5: Neutrons cross section of the MNIST CNN using matrix multiplication



But then again, the cross section alone doesn't tell the whole story. In our experiments, the ABFT has managed to correct less than 4% of the errors. On top of that, Fig. 7.6 shows us that using ABFT on the convolutional layers also adds a lot of overhead in terms of clock cycles, and by calculating the MEBF (Fig. 7.7), we see that we should in fact opt for the unhardened version.

Figure 7.6: Clock cycles taken by the MNIST CNN using matrix multiplication to process one input image

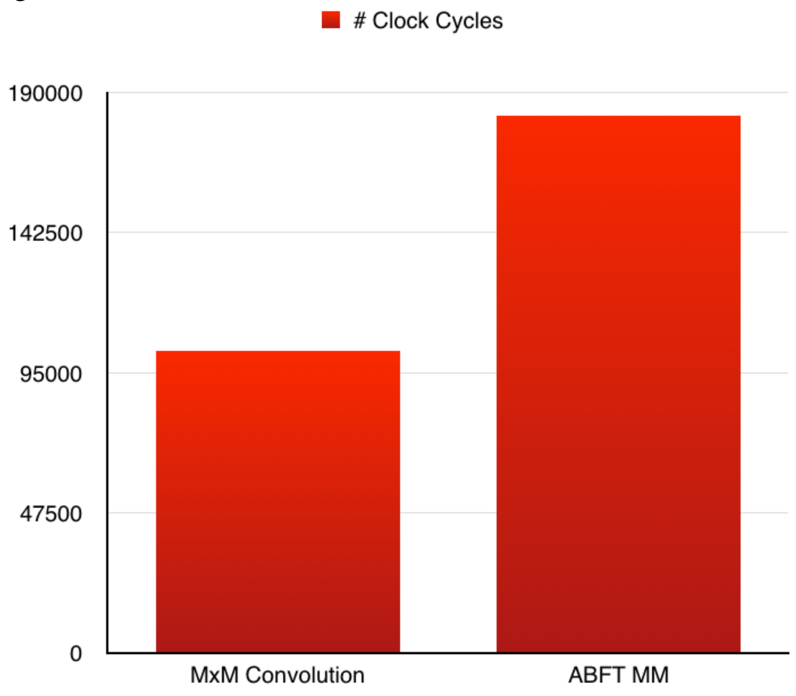
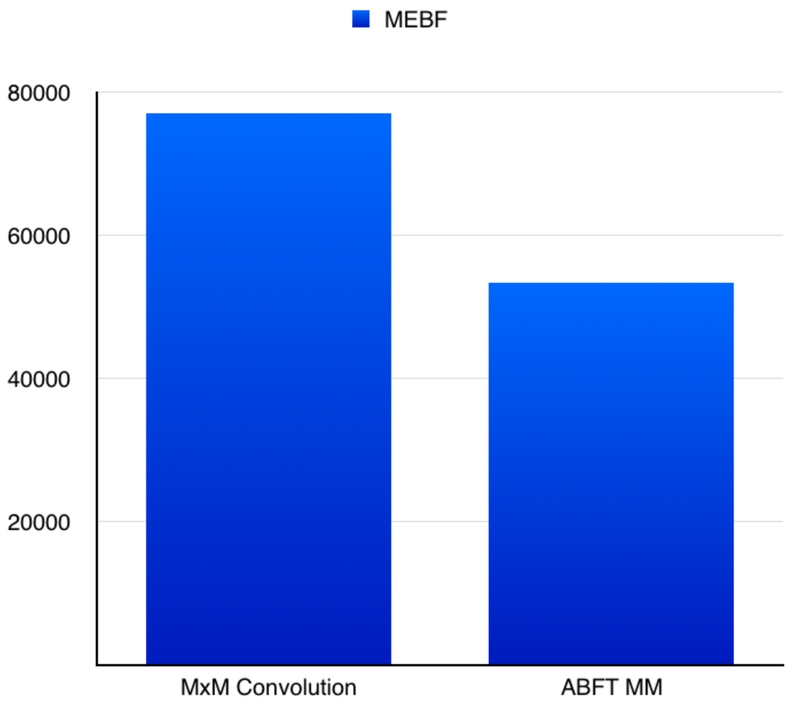


Figure 7.7: MEBF of the MNIST CNN using matrix multiplication



7.1.2 SmartPool

As explained in Chapter 2.3, pooling layers are fundamental for CNNs as they reduce the amount of information that propagates through the network. In terms of reliability, pooling could help in masking radiation-induced faults as some of the values (possibly corrupted) are not passed along to the following layer.

We have decided to test an ABFT technique that applies to the pooling layers using the "maximum" operation. This technique is called *SmartPool* and it consists of adding a few extra steps to the maxpooling operation: after determining the maximum value in a cluster, we compare such value to a pre-determined threshold. If the value exceeds the threshold, we discard it and select the second highest value instead. The threshold is calculated beforehand (when designing and testing the network, without radiation) considering all the possible correct values that are propagated by each pool layer. SmartPool was proved to be very effective in GPUs (SANTOS et al., 2017), and as the associated overhead is not huge (in our case, it uses approximately 19% more resources than the design with regular maxpool), we've decided to test it on FPGAs. Fig. 7.8 shows resource utilization and Fig. 7.9 shows the cross section of the MNIST CNN using smartpool.

As we can see from Fig 7.9, this time around, the cross section actually increased (even if slightly) by adding smartpool to the pooling layers of the CNN. Additionally, if we look at the overhead in terms of clock cycles (Fig. 7.10) and the calculated MEBF (Fig. 7.11), we come to the conclusion, as we did with the fault tolerant matrix multiplication, that smartpool is also not worth it in FPGAs.

This, of course, is a not an optimal result, but it might point towards the argument that the most critical portion of CNNs is in fact the full connected MLP classifier. In Chapter 7.2, with use this finding, alongside the architectural vulnerability data from our fault injection, in an attempt to address such issue.

Figure 7.8: Resource utilization by the MNIST CNN using smartpool

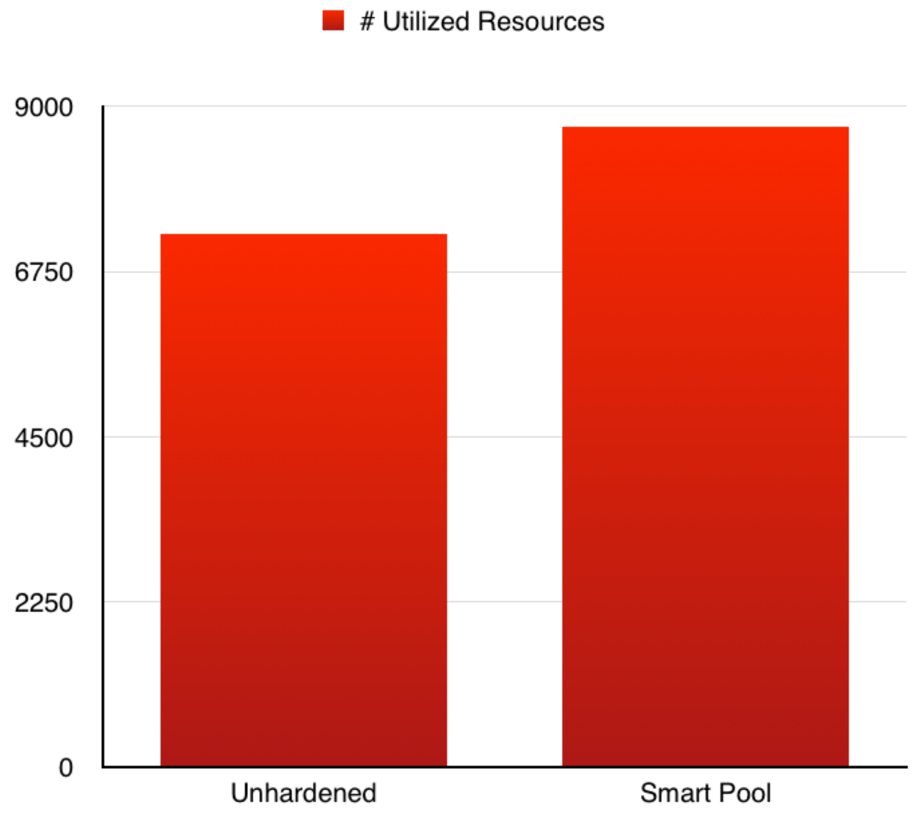


Figure 7.9: Neutrons cross section of the MNIST CNN using smartpool

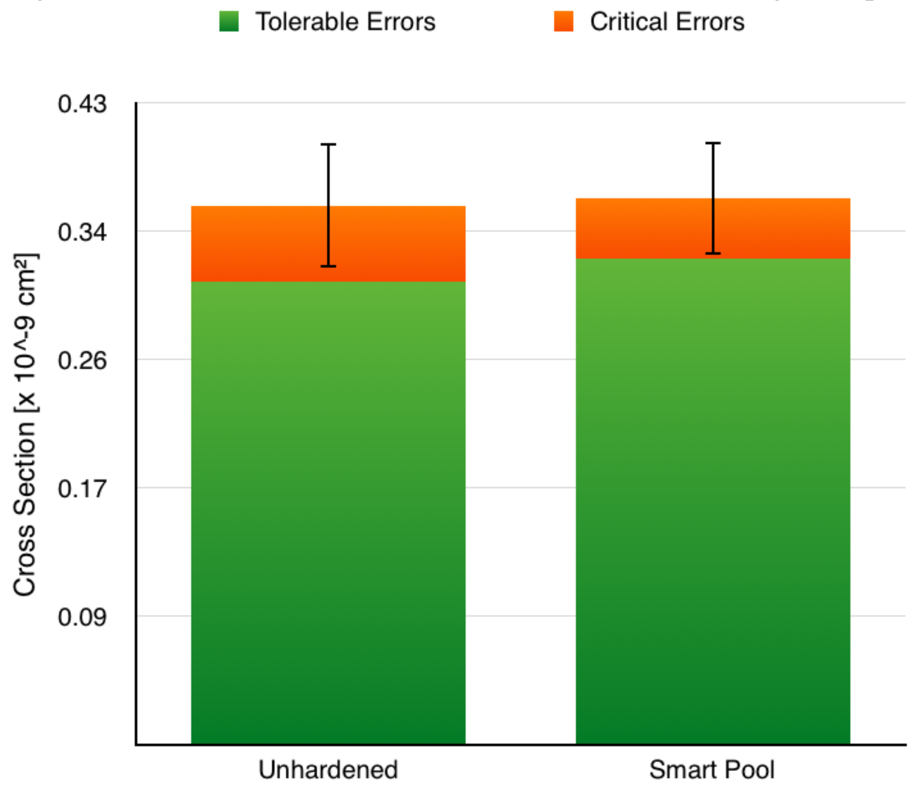


Figure 7.10: Clock cycles taken by the MNIST CNN using smartpool to process one input image

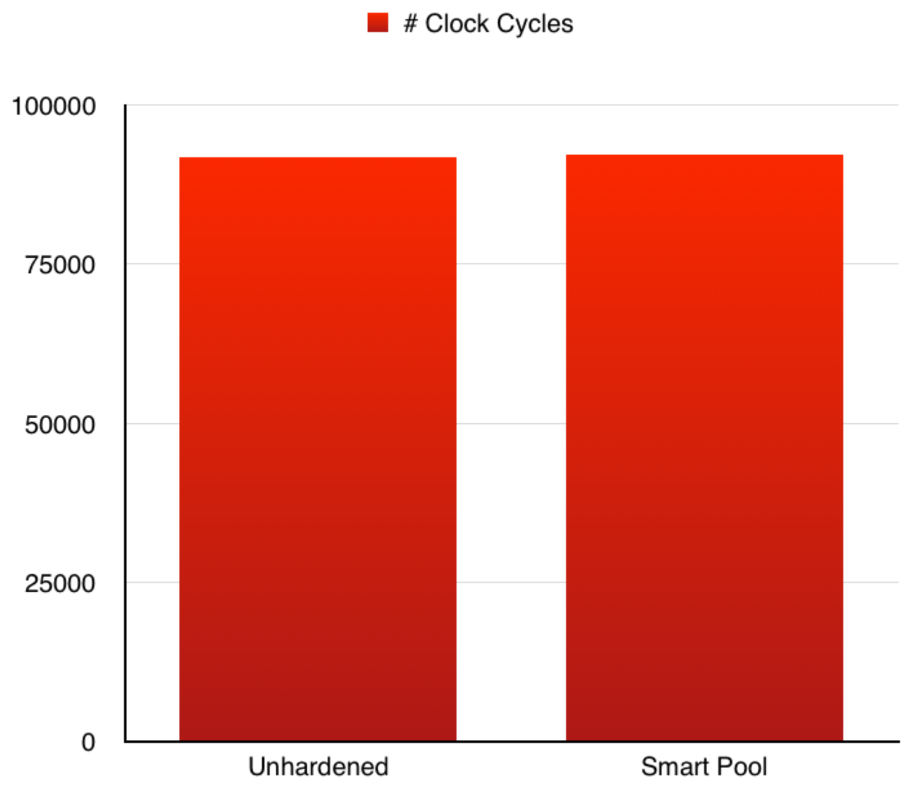
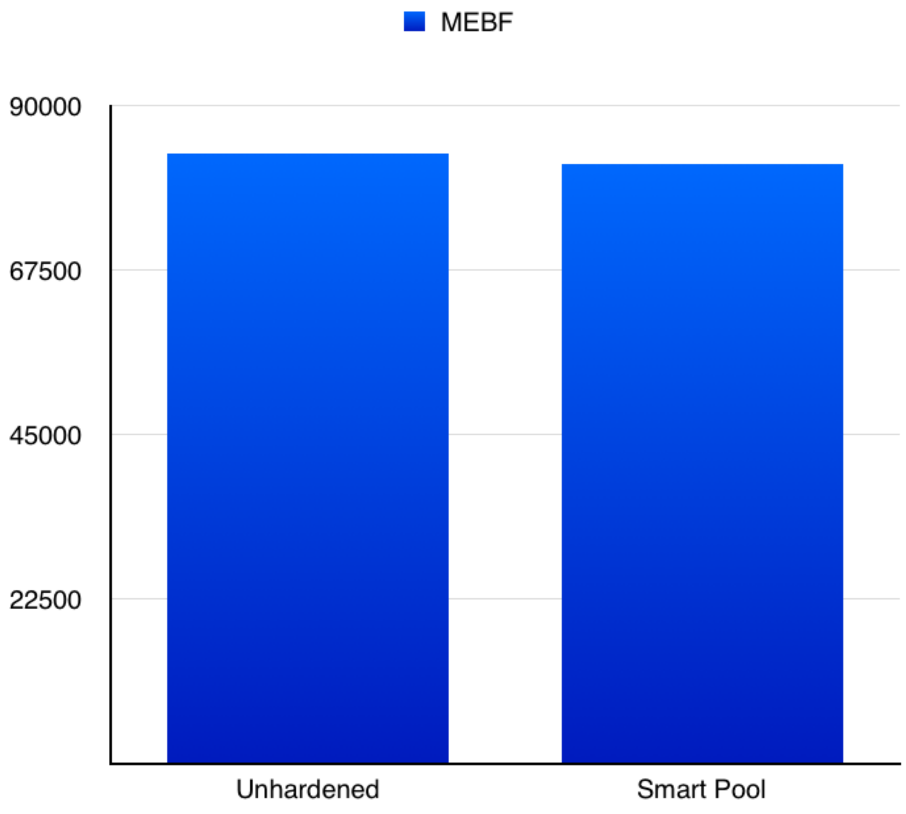


Figure 7.11: MEBF of the MNIST CNN using smartpool



7.2 Selective Redundancy

Motivated by the frustrating results obtained with ABFT strategies, we have decided to change the hardening paradigm and go for modular redundancy as a way to harden our neural networks. Based on our fault-injection campaigns (showcased in Chapter 6.2), we conclude that the criticality of a given layer in a neural network is determined by its function, size, and position. In both case-studies, there is a considerable gap in vulnerability across the different layers, which motivates the idea of selective hardening. What we mean by selective hardening is that we choose particular portions of the ANNs to harden with redundancy. We could, hypothetically, triplicate the entire neural network, without considering the variance in vulnerability from the different layers, but the additional overhead from this strategy may not be worth its benefit. Furthermore, we have seen that full TMR is not always feasible in terms of resources on the FPGAs, given that state-of-the-art ANNs require a lot of computing power. In order to evaluate and compare different hardening strategies, we introduce the concept of *Hardening Efficiency*, which is essentially the symbolic trade-off between benefit and cost of a given hardening solution.

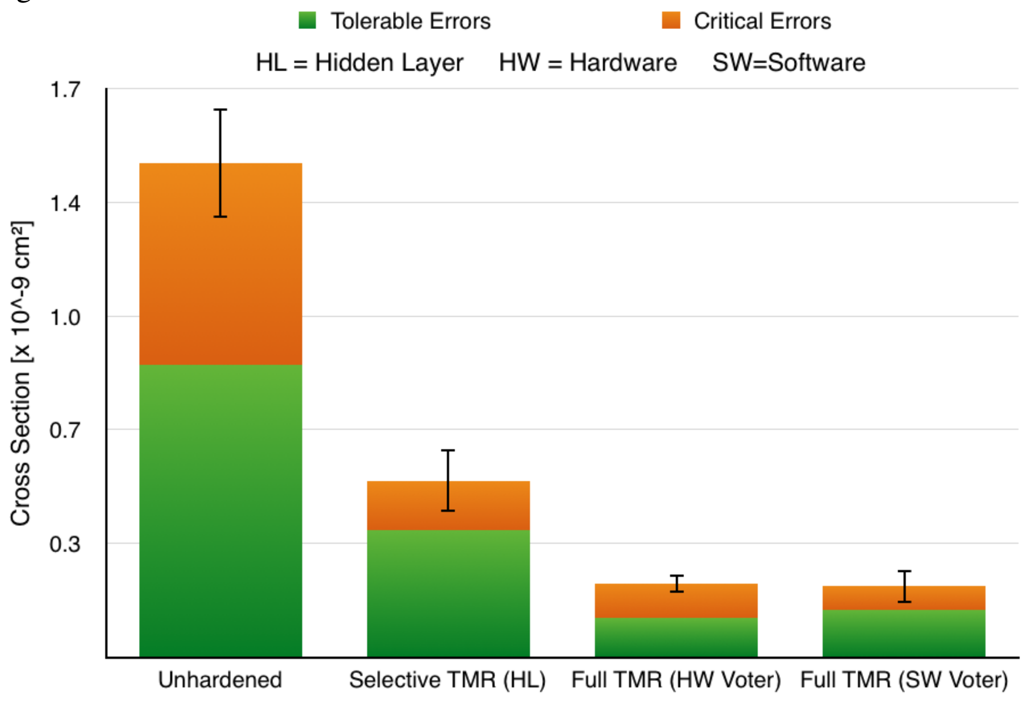
7.2.1 Iris Flower

We have designed four configurations of the Iris Flower ANN to be tested on the Zynq-7000 device:

- **Unprotected:** the plain ANN, without any hardening
- **Selective TMR:** the ANN with a triplication of the Hidden Layer (HL), alone
- **Full TMR (HW Voter):** the ANN fully triplicated, with the voter implemented in the PL part (FPGA).
- **Full TMR (SW Voter):** the ANN fully triplicated, with the voter implemented in the PS part (ARM).

We initially chose to evaluate two voting mechanisms to see if there was an advantage in terms of reliability, by delegating the task to the processor (consequently, reducing resource utilization in the FPGA). But, as we can see in Fig. 7.12, there was a negligible difference between the two designs, so we have decided not to test the SW Voter version with the MNIST CNN.

Figure 7.12: Neutrons cross section of the Iris Flower ANN using four different hardening configurations



The Iris Flower ANN implemented on the Zynq-7000 was tested at LANSCE in October 2017. We tested the device for 40 hours, accumulating a total fluence of more than $1.6x10^{11} neutrons/cm^2$ and gathering more than 400 events. We can clearly perceive, from Fig. 7.12, that the probability of error occurrence for the selective hardened version decreases by 64.2% with respect to the unhardened version. Similarly the percentage reductions considering the Full TMR configurations are 85.1% and 85.6% for the voter implemented in the FPGA and in the ARM processor, respectively.

The Full TMR versions outperform the Selective TMR configurations in terms of pure reliability, which is to be expected. Fig. 7.13 shows the number of used resources (LUTs, DSPs, FFs) for all the different designs, along with the percentage of masked faults (meaning, from all the faults that occurred, what percentage the TMR successfully corrected). As shown, the overhead of a Full TMR is much bigger (almost 2x) than the one of the Selective TMR. In other words, with a 45% increase in resource usage our selective hardening achieves 68% of fault masking, while traditional TMR approaches require around a 200% increase in area to reach near 100% rate of fault masking (94% in this case). If we divide the percentage of masked faults (benefit) by the number of utilized resources (cost), we get another metric called *Hardening Efficiency*, as we mentioned in the beginning of the chapter. We have plotted this in Fig. 7.14, where we can see that our selective hardening approach is around 25% more efficient than traditional full TMR.

Figure 7.13: Masked faults percentage and resource utilization of the four different hardening configurations of the Iris Flower ANN

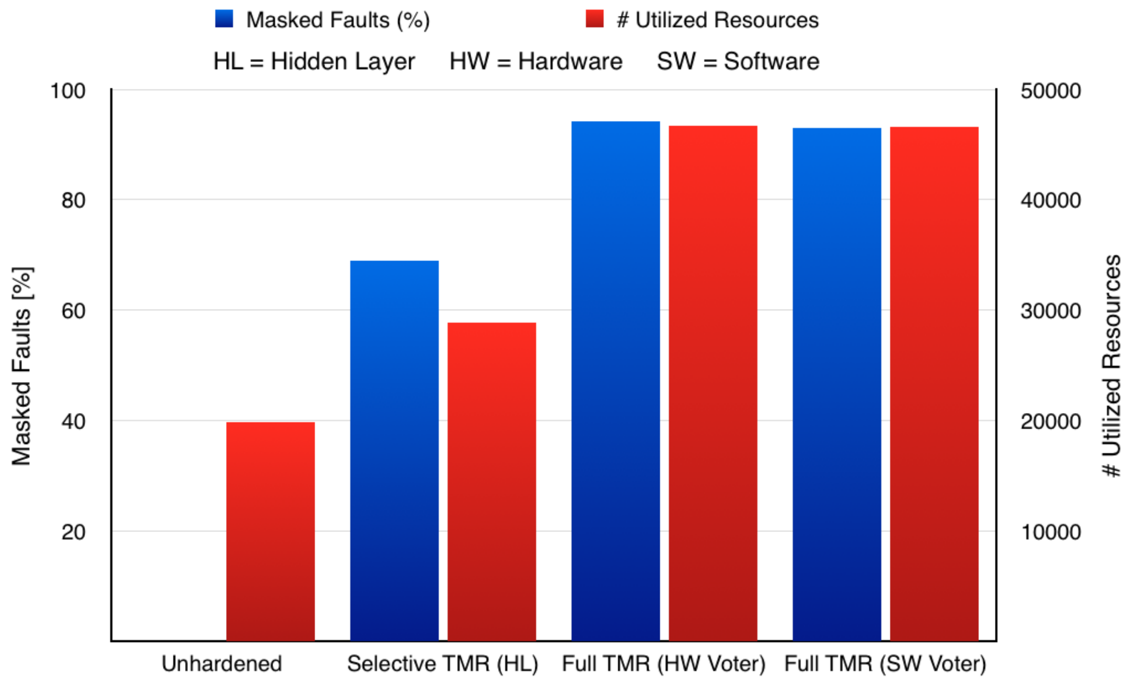
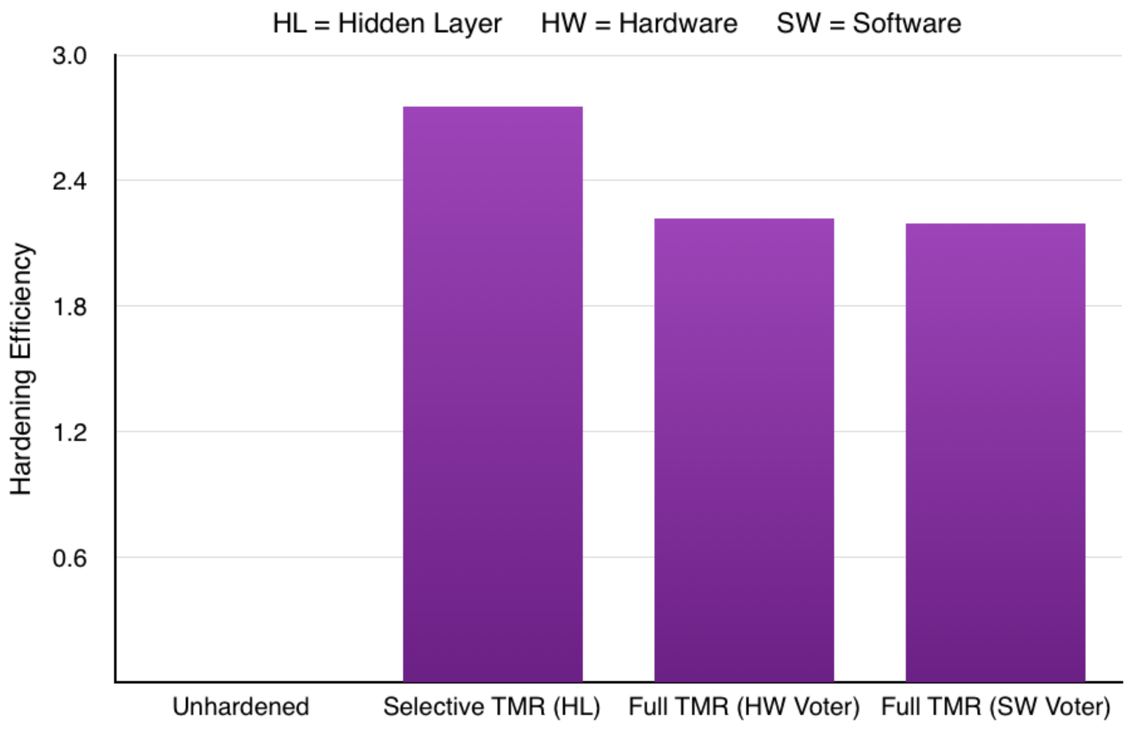


Figure 7.14: Percentage of masked faults divided by resource utilization of the four different hardening configurations of the Iris Flower ANN.

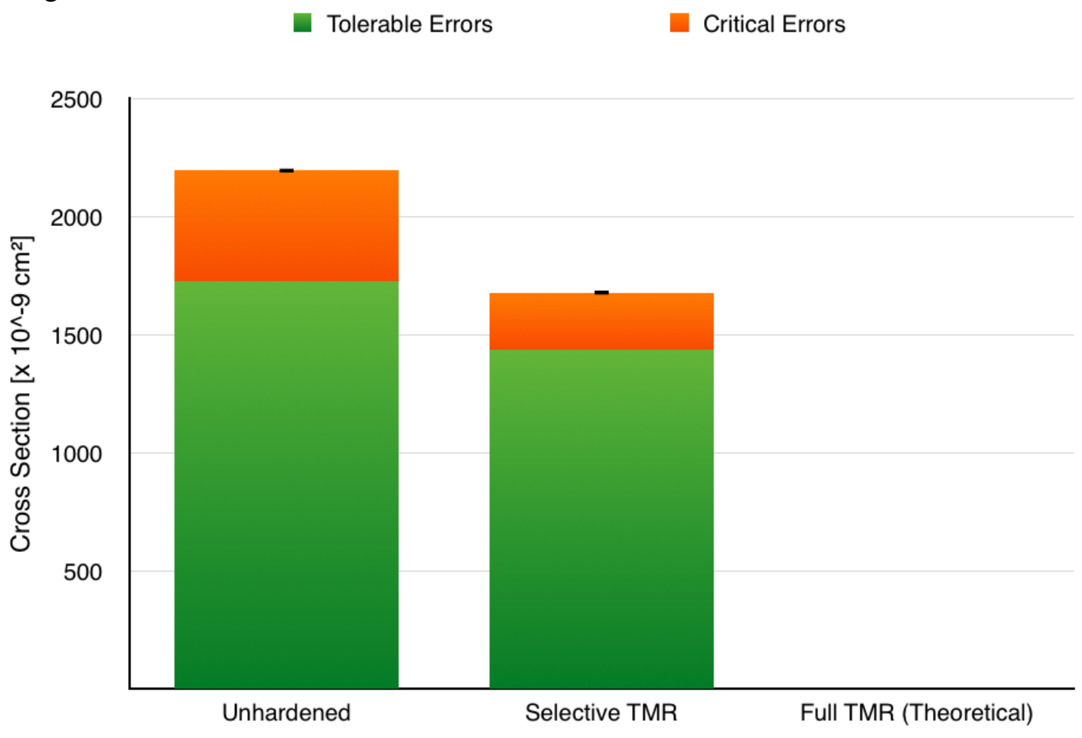


7.2.2 MNIST

For the MNIST CNN, we have seen in our fault injection campaign that the very last layer of the network (called *IP2*) seemed to be the most vulnerable. Moreover the strategies applied to the convolution part of the network were not very effective, suggesting that errors in convolution hardly affect classification. Thus, we have decided to apply our selective TMR strategy to the IP2 layer. We then tested two versions of the MNIST CNN in our beam experiments: the unhardened neural network and the selective TMR one (with the IP2 layer fully triplicated). We did not test the Full TMR version as we did with the Iris Flower ANN, simply because it was not feasible resource-wise, meaning that there weren't enough available LUTs in the Zynq Ultrascale+ device to fully triplicate the MNIST CNN. This fact is showcased on the very last line of Table I, and it reinforces the importance of selective hardening as a more efficient use of additional resources.

The MNIST CNN implemented on the Zynq Ultrascale+ was tested at ChipIR in June 2018. We tested the device for 20 hours, accumulating a total fluence of more than $3.0x10^{11} neutrons/cm^2$ and gathering more than 1.5 million events. Fig. 7.15 shows the neutron cross section of the MNIST CNN. We can clearly see that the Selective TMR reduced the probability of occurrence of both tolerable and critical errors (precisely, 48% reduction of the latter).

Figure 7.15: Neutrons cross section of the MNIST CNN using three different hardening configurations.



In addition to that, we have also put Fig. 7.16 here to highlight the percentage of masked faults by each design, along with their absolute number of utilized resources. Note that our selective hardening approach achieves around 40% fault-masking with a marginal overhead of only 8%. Also note that, as we mentioned before, it wasn't possible to test the Full TMR version, but we completed the graph with theoretical values (near 100% fault-masking, with near 200% overhead). Once again, if we look at the hardening efficiency graph (Fig. 7.17), we can clearly perceive that our proposed selective hardening approach offers the better trade-off, leaving the traditional TMR behind (even if this is a calculated theoretical efficiency, that considers perfect fault-masking using Full TMR).

Figure 7.16: Masked faults percentage and resource utilization of the three different hardening configurations of the MNIST CNN

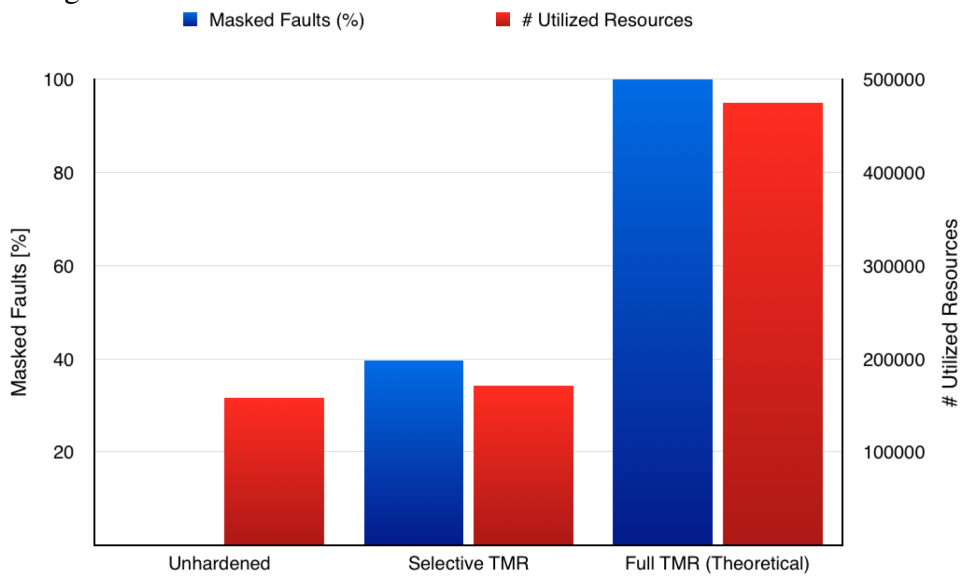
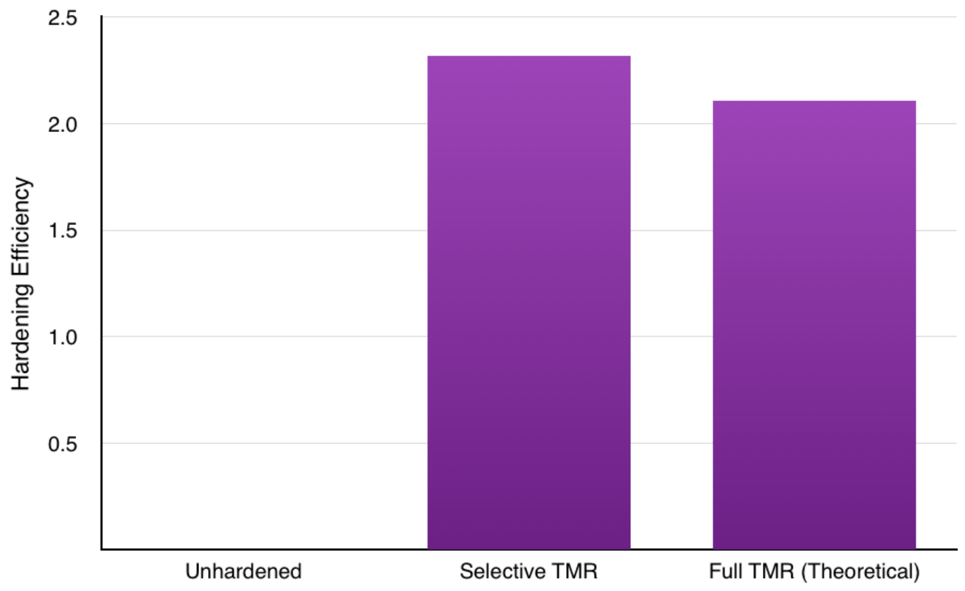


Figure 7.17: Percentage of masked faults divided by resource utilization of the three different hardening configurations of the MNIST CNN.



7.3 Discussion

We've seen that ABFT techniques on convolutional and pooling layers of CNNs are not effective in FPGAs, even though they are in GPUs. The reader might be wondering why we didn't try to apply some kind of ABFT to the *Inner Product* and *ReLU* layers of MNIST CNN. Inner Product layers are basically matrix multiplications, so they would be a candidate for the ABFT MxM, if it wasn't for the fact that one of the matrices involved in Inner Product have one dimension of size 1. This means that the additional line in that input matrix does not add new information, but just duplicates the existing. ReLU, on the other hand, doesn't really need any ABFT, since it already masks 50% of all input errors by definition (as we can see on Fig 2.8, of Chapter 2.3), and as we demonstrated with our fault injection experiment (in Chapter 6.2).

Furthermore, we've seen that selectively triplicating the most vulnerable layer of the neural network proved to be an efficient hardening strategy on both case-studies. This suggests that our proposed selective hardening might bring the best hardening efficiency in most neural networks, despite their size and complexity of topology.

8 CONCLUSIONS & FUTURE WORK

We have proven that, due to the approximate computing nature of classification ANNs, not every error is to be considered critical. Furthermore, we also found out that the critical errors are the minority in FPGAs implementing ANNs, given that we saw this across all case-studies and experiments.

Beyond that, with the insights from our fault injection campaigns, we have seen that the different layers on ANNs (be it a simple MLP, or a more complex ANN), have considerably different levels of vulnerability.

As an attempt to improve the reliability of our ANNs, first we tried to apply ABFT techniques to the algorithms involved in the different types of layers of our case-study CNN. This approach has been proved to be successful in other parallel devices, such as GPUs, so we wanted to verify if the outcome was similar in FPGAs. Unfortunately, results were not exciting, as the two tested ABFT strategies showed little to no improvements in terms of reliability, while adding a lot of overhead in terms of resources and timing.

On this scenario, we have decided to change the hardening paradigm, by trying to improve the reliability on ANNs using modular redundancy (thus, at an architectural level). Given that the overhead of traditional techniques such as TMR is always really high, and sometimes not even feasible (resource-wise) to implement in FPGAs, we proposed a selective approach. The idea behind our strategy was to use the information about layer vulnerability (that we already had from fault injection), and to protect (via triplication) only the most critical parts of our ANNs.

With new beam tests, we successfully validated the theory behind our selective hardening approach: on the small MLP network, we achieved 68% fault-masking with a 45% overhead, while on the larger CNN, we reached 40% fault-masking with only 8% overhead. Whenever we compared the efficiency of our hardening strategy to traditional full TMR, we saw that our strategy came out on top with both neural networks, even considering theoretical numbers for the full TMR (100% fault-masking with 200% overhead).

If we combine the facts that ABFT wasn't effective at the initial layers of the CNN (responsible for feature extraction), and that selective triplication proved to be effective in full connected layers, it is reasonable to conclude that, not only each layer has its own level of vulnerability, but the two main parts of a CNN impact the overall reliability in different ways.

As such, one of our intents for future work, is to perform new fault injection campaigns, but this time augmenting the level of granularity to only two parts: feature extraction layers and full connected layers. We believe we will be able to prove our mentioned hypothesis, and then start to think of new creative ways of hardening neural networks.

In addition to that, we also intend to explore more aspects of the training phase. We have already experimented with a technique similar to weight pruning (which consists in finding and eliminating unnecessary weights in a trained model, in order to improve performance and reduce memory requirements), in which we have managed to prune around 57% of the weights in our MNIST CNN, without significantly reducing accuracy. The most exciting thing here is that most of the pruned weights are from the full connected layers. This makes perfect sense, since convolutional layers re-use their weights in order to explore spatial and temporal invariance, while fully-connected layers only use each weight once (ultimately needing a much bigger set of them). We believe that reducing the amount of weights in the MLP part of CNNs can contribute to a considerable increase in terms of performance, and could also improve overall reliability, even though it may increase the vulnerability of this specific part of the networks.

Finally, we also want to make a deeper comparison between FPGAs and GPUs. We want to determine how related or unrelated are the fault model and the vulnerability level of each layer. With these two pieces of information, we want to build a portfolio of viable hardening strategies for both of them, thus, for generic neural networks in parallel devices.

REFERENCES

ABDELOUAHAB, K. et al. Tactics to directly map cnn graphs on embedded fpgas. **IEEE Embedded Systems Letters**, p. 1–4, 2017. ISSN 19430663. Available from Internet: <http://ieeexplore.ieee.org/document/8015156/>.

ABRACADABRA. **Working with CNNs in practice**. 2017. Available from Internet: <https://ewanlee.github.io/2017/04/10/cs231n-Lecture-11-Recap/>.

ALLEN, G. R. et al. 2017 Compendium of Recent Test Results of Single Event Effects Conducted by the Jet Propulsion Laboratorys Radiation Effects Group. **2017 IEEE Radiation Effects Data Workshop (REDW)**, 2017.

ALTERA. **Cyclone V FPGAs & SoCs**. 2017. Available from Internet: <https://www.altera.com/products/fpga/cyclone-series/cyclone-v/overview.html>.

AMIN, S. U. et al. Genetic neural network based data mining in prediction of heart disease using risk factors. **2013 IEEE Conference On Information And Communication Technologies**, 2013.

ANDERSON, E. The Irises of the Gaspe Peninsula. **Bulletin of the American Iris Society**, v. 59, p. 2–5, 1935.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, 2004.

BAUMANN, R. Radiation-induced soft errors in advanced semiconductor technologies. **IEEE Transactions on Device and Materials Reliability**, v. 5, n. 3, p. 305–316, 2005.

BERNARDESCHI, C. et al. SRAM-Based FPGA Systems for Safety-Critical Applications: A Survey on Design Standards and Proposed Methodologies. **Journal of Computer Science and Technology**, v. 30, n. 2, p. 373–390, 2015.

BRAUN, C.; HALDER, S.; WUNDERLICH, H. J. A-abft: Autonomous algorithm-based fault tolerance for matrix multiplications on graphics processing units. **2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**, 2014.

CAMBRIDGE SPARK. **Deep learning for complete beginners: convolutional neural networks with keras**. 2017. Available from Internet: <https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html>.

CAZZANIGA, C.; FROST, C. D. Progress of the scientific commissioning of a fast neutron beamline for chip irradiation. **Journal of Physics: Conference Series**, v. 1021, p. 012037, 2018.

CHELLAPILLA, K.; PURI, S.; SIMARD, P. CHigh Performance Convolutional Neural Networks for Document Processing. **Tenth International Workshop on Frontiers in Handwriting Recognition**, 2006.

DATA SCIENCE-STACK EXCHANGE. . 2017. Available from Internet: <https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size/23186>.

DENG, L. The mnist database of handwritten digit images for machine learning research [best of the web]. **IEEE Signal Processing Magazine**, v. 29, n. 6, p. 141–142, 2012.

DING, R. et al. LightNN: Filling the Gap between Conventional Deep Neural Networks and Binarized Networks. **Proceedings of the on Great Lakes Symposium on VLSI 2017 - GLSVLSI 17**, 2017.

ELECTREK. **Elon Musk on Tesla fully autonomous car**. 2016. Available from Internet: <https://electrek.co/2016/08/03/elon-musk-tesla-fully-autonomous-car-blows-mind>.

FISHER, R. A. The Use Of Multiple Measurements In Taxonomic Problems. **Annals of Eugenics**, v. 7, n. 2, p. 179–188, 1936.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep Sparse Rectifier Neural Networks. In: GORDON, G.; DUNSON, D.; DUDíK, M. (Ed.). **Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics**. Fort Lauderdale, FL, USA: PMLR, 2011. (Proceedings of Machine Learning Research, v. 15), p. 315–323.

HAHNLOSER, R. H. R. et al. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. **Nature**, v. 408, n. 6815, p. 1012–1012, 2000.

HE, C. et al. A novel soc architecture on fpga for ultra fast face detection. **2009 IEEE International Conference on Computer Design**, 2009.

HUANG, K.-H.; ABRAHAM, J. Algorithm-Based Fault Tolerance for Matrix Operations. **Computers, IEEE Transactions on**, C-33, n. 6, p. 518–528, June 1984. ISSN 0018-9340.

IEC. **Functional Safety**. 2017. Available from Internet: <http://www.iec.ch/functionalsafety/faq-ed2/page5.htm>.

IEEE. In: **IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries**. New York, NY, USA: IEEE, 1990. Available from Internet: <https://ieeexplore.ieee.org/document/182763/>.

JACOBS, A.; CIESLEWSKI, G.; GEORGE, A. D. Overhead and reliability analysis of algorithm-based fault tolerance in fpga systems. **22nd International Conference on Field Programmable Logic and Applications (FPL)**, 2012.

JEDEC. **Rep. JESD89A, JEDEC Standard**. 2006. Available from Internet: <https://www.jedec.org/sites/default/files/docs/jesd89a.pdf>.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998.

MEROLLA, P. A. et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. **Science**, v. 345, n. 6197, p. 668–673, Jul 2014.

MICROSEMI. **SmartFusion SoC FPGAs**. 2017. Available from Internet: <http://www.microsemi.com/products/fpga-soc/soc-fpga/smartfusion>.

MILITARY.COM. **Air Force to Allow Enlisted Airmen to Fly Global Hawk Drones**. 2015. Available from Internet: <https://www.military.com/daily-news/2015/12/17/air-force-to-allow-enlisted-airmen-to-fly-global-hawk-drones.html>.

MINSKY, M.; PAPERT, S.; LEON, B. **Perceptrons: an introduction to computational geometry**. [S.l.]: The MIT Press, 1969.

MOORE, G. Cramming More Components Onto Integrated Circuits. **Proceedings of the IEEE**, v. 86, n. 1, p. 82–85, 1965.

NEAGOE, V.-E. et al. A concurrent neural network approach to pedestrian detection in thermal imagery. **2012 9th International Conference on Communications (COMM)**, 2012.

PRATT, B. et al. Fine-Grain SEU Mitigation for FPGAs Using Partial TMR. **IEEE Transactions on Nuclear Science**, v. 55, n. 4, p. 2274–2280, 2008.

QUARTZ. **Google's AI won the game Go by defying millennia of basic human instinct**. 2016. Available from Internet: <https://qz.com/639952/>.

RECH, P. et al. An efficient and experimentally tuned software-based hardening strategy for matrix multiplication on gpus. **IEEE Transactions on Nuclear Science**, IEEE, v. 60, n. 4, p. 2797–2804, 2013.

RECH, P. et al. Impact of gpus parallelism management on safety-critical and hpc applications reliability. **2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**, 2014.

RIZVI, S. T. H.; CABODI, G.; FRANCINI, G. Gpu-only unified convmm layer for neural classifiers. **2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)**, 2017.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, n. 6, p. 386–408, 1958.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. [S.l.]: Prentice Hall, 2010.

SAMUEL, A. L. Some Studies in Machine Learning Using the Game of Checkers. **IBM Journal of Research and Development**, v. 3, n. 3, p. 210–229, 1959.

SANTOS, F. F. D. et al. Evaluation and mitigation of soft-errors in neural network-based object detection in three gpu architectures. **2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)**, 2017.

TAMBARA, L. A. **Analyzing the Impact of Radiation-induced Failures in All Programmable System-on-Chip Devices**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2017.

TAMBARA, L. A. et al. Analyzing the impact of radiation-induced failures in programmable socs. **IEEE Transactions on Nuclear Science**, v. 63, n. 4, p. 2217–2224, 2016.

TONFAT, J. et al. Analyzing the Effectiveness of a Frame-Level Redundancy Scrubbing Technique for SRAM-based FPGAs. **IEEE Transactions on Nuclear Science**, v. 62, n. 6, p. 3080–3087, 2015.

TONFAT, J. et al. Method to analyze the susceptibility of hls designs in sram-based fpgas under soft errors. **Lecture Notes in Computer Science Applied Reconfigurable Computing**, p. 132–143, 2016.

VALUEWALK. **NASA Plans A ‘Mars Helicopter’ Mission**. 2018. Available from Internet: <https://www.valuewalk.com/2018/05/mars-helicopter-explore-red-planet/>.

VASUDEVAN, A.; ANDERSON, A.; GREGG, D. Parallel multi channel convolution using general matrix multiplication. **2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)**, 2017.

WIRTHLIN, M. High-reliability fpga-based systems: Space, high-energy physics, and beyond. **Proceedings of the IEEE**, v. 103, n. 3, p. 379–389, 2015.

XILINX. **UG474 - 7 Series FPGAs Configurable Logic Block**. 2016. Available from Internet: <https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf>.

XILINX. **Xilinx SoC, MPSoC and RFSoC Feature Summary**. 2017. Available from Internet: <http://www.xilinx.com/products/silicon-devices/soc/>.

XILINX. **Zynq-7000 SoC Product Advantages**. 2018. Available from Internet: <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>.

XILINX. **Zynq Ultrascale+ SoC Product Advantages**. 2018. Available from Internet: <https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html>.

ZHOU, Y.; REDKAR, S.; HUANG, X. Deep learning binary neural network on an fpga. **2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)**, 2017.