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

ISRAEL DA COSTA LOPES

Convolutional Neural Network Reliability on an APSoC platform - a traffic-sign recognition case study

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

Advisor: Prof. Dr. Altamiro Amadeu Susin

Porto Alegre
November 2017

CIP — CATALOGING-IN-PUBLICATION

Lopes, Israel da Costa

Convolutional Neural Network Reliability on an APSoC platform - a traffic-sign recognition case study / Israel da Costa Lopes. – Porto Alegre: PGMICRO da UFRGS, 2017.

94 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2017. Advisor: Altamiro Amadeu Susin.

1. Deep learning. 2. Traffic-sign recognition. 3. Soft errors. 4. System-on-Chip. I. Susin, Altamiro Amadeu. II. Título.

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

ACKNOWLEDGEMENTS

Agradeço a Deus e minha família que me deram todo o suporte emocional, financeiro e proteção. Ao meu orientador Altamiro Susin que me incentivou e me deu as diretrizes necessárias para a conclusão do meu mestrado. A Professora Fernanda que me introduziu aos conceitos de tolerância a radiação, análise de confiabiliadade e que me apoiou em trabalhos de campo para experimentos de radiação. Aos meus colegas aos quais colaboraram muito para meu crescimento acadêmico e profissional. A CAPES por prover a bolsa de mestrado permitindo que eu pudesse me dedicar exclusivamente ao mestrado. E a todos os técnicos e funcionários da UFRGS que tornaram tudo isso possível.

ABSTRACT

Deep learning has a plethora of applications in computer vision, speech recognition, natural language processing and other applications of commercial interest. Computer vision, in turn, has many applications in distinct areas, ranging from entertainment applications to relevant and critical applications. Face recognition and manipulation (Snapchat), and object description in pictures (OneDrive) are examples of entertainment applications. Industrial inspection, medical diagnostics, object recognition in images captured by satellites (used in rescue and defense missions), autonomous cars and Advanced Driver-Assistance System (ADAS) are examples of relevant and critical applications. Some of the most important integrated circuit companies around the world, such as Xilinx, Intel and Nvidia are waging in dedicated platforms for accelerating the training and deployment of deep learning and other computer vision algorithms for autonomous cars and ADAS due to their high computational requirement. Thus, implementing a deep learning system that achieves high performance with low area utilization and power consumption costs is a big challenge. Besides, electronic equipment for automotive industry must be reliable even under radiation effects, manufacturing defects and aging effects, inasmuch as if a system failure occurs, a car accident can happen. Thus, a Convolutional Neural Network (CNN) VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL) automatic generator was developed to reduce the design time associated to the implementation of deep learning algorithms in hardware. As a case study, a CNN was trained by the Convolutional Architecture for Fast Feature Embedding (Caffe) framework, in order to classify 6 traffic-sign classes, achieving an average accuracy of about 89.8% on the German Traffic-Sign Recognition Benchmark (GTSRB) dataset, which contains traffic-signs images in complex scenarios. This CNN was implemented on a Zynq-7000 All-Programmable System-on-Chip (APSoC), achieving about 313 Frames Per Second (FPS) on 32x32-normalized images, with the APSoC consuming only 2.057 W, while an embedded Graphics Processing Unit (GPU), in its minimum operation mode, consumes 10W. The proposed CNN reliability was investigated by random piled-up fault injection by emulation in the Programming Logic (PL) configuration bits of the APSoC, achieving 80.5% of reliability under Single-Bit-Upset (SBU) where both critical Silent Data Corruptions (SDCs) and time-outs were considered. Regarding the multiple faults, the proposed CNN reliability exponentially decreases with the number of piled-up faults. Hence, the proposed CNN reliability must be increased by using hardening techniques during the design

flow.

Keywords: Deep learning. Traffic-sign recognition. Soft errors. System-on-Chip.

Confiabilidade de uma Rede Neural Convolucional em uma plataforma APSoC - um caso de estudo para reconhecimento de placas de trânsito

RESUMO

O aprendizado profundo tem inúmeras aplicações na visão computacional, reconhecimento de fala, processamento de linguagem natural e outras aplicações de interesse comercial. A visão computacional, por sua vez, possui muitas aplicações em áreas distintas, indo desde o entretenimento à aplicações relevantes e críticas. O reconhecimento e manipulação de faces (Snapchat), e a descrição de objetos em fotos (OneDrive) são exemplos de aplicações no entretenimento. Ao passo que, a inspeção industrial, o diagnóstico médico, o reconhecimento de objetos em imagens capturadas por satélites (usadas em missões de resgate e defesa), os carros autônomos e o Sistema Avançado de Auxílio ao Motorista (SAAM) são exemplos de aplicações relevantes e críticas. Algumas das empresas de circuitos integrados mais importantes do mundo, como Xilinx, Intel e Nvidia estão apostando em plataformas dedicadas para acelerar o treinamento e a implementação de algoritmos de aprendizado profundo e outras alternativas de visão computacional para carros autônomos e SAAM devido às suas altas necessidades computacionais. Assim, implementar sistemas de aprendizado profundo que alcançam alto desempenho com o custo de baixa utilização de área e dissipação de potência é um grande desafio. Além do mais, os circuitos eletrônicos para a indústria automotiva devem ser confiáveis mesmo sob efeitos da radiação, defeitos de fabricação e efeitos do envelhecimento. Assim, um gerador automático de *VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL)* para Redes Neurais Convolucionais (RNC) foi desenvolvido para reduzir o tempo associado a implementação de algoritmos de aprendizado profundo em *hardware*. Como estudo de caso, uma RNC foi treinada pela ferramenta *Convolutional Architecture for Fast Feature Embedding (Caffe)*, de modo a classificar 6 classes de placas de trânsito, alcançando uma precisão de cerca de 89,8% no conjunto de dados *German Traffic-Sign Recognition Benchmark (GTSRB)*, que contém imagens de placas de trânsito em cenários complexos. Essa RNC foi implementada num *All-Programmable System-on-Chip (APSoC)* Zynq-7000, resultando em 313 Frames Por Segundo (FPS) em imagens normalizadas para 32x32, com o *APSoC* dissipando uma potência de somente 2.057 W, enquanto uma *Graphics Processing Unit (GPU)* embarcada, em seu modo de operação mínimo, dissipa 10 W. A confiabilidade da RNC proposta foi investigada por injeções

de falhas acumuladas e aleatórias por emulação nos *bits* de configuração da Lógica Programável (LP) do APSoC, alcançando uma confiabilidade de 80,5% sob *Single-Bit-Upset (SBU)* onde foram considerados ambos os Dados Corrompidos Silenciosos (DCSs) críticos e os casos em que o sistema não respondeu no tempo esperado (*time-outs*). Em relação às falhas múltiplas, a confiabilidade da RNC decresce exponencialmente com o número de falhas acumuladas. Em vista disso, a confiabilidade da RNC proposta deve ser aumentada através do uso de técnicas de proteção durante o fluxo de projeto.

Palavras-chave: Aprendizado profundo. Reconhecimento de placas de trânsito. Erros *Soft*, Sistema-em-Chip..

LIST OF ABBREVIATIONS AND ACRONYMS

ADAS	Advanced Driver-Assistance System
AI	Artificial intelligence
AVF	Architectural Vulnerability Factor
SDC	Silent Data Corruption
ACE	Architecturally Correct Execution
CNN	Convolutional Neural Network
APSoC	All-Programmable System-on-Chip
MPSoC	Multiprocessor System-on-Chip
FinFET	Fin-Field Effect Transistor
SoC	System-on-Chip
PS	Processing System
PL	Programming Logic
DUT	Design Under Test
AXI	Advanced eXtensible Interface
ICAP	Internal Configuration Access Port
GPIO	General Purpose Input/Output
BRAM	Block Random Access Memory
SRAM	Static Random Access Memory
ROM	Read Only Memory
DDR	Double Data Rate
LUT	Look Up Table
FF	Flip-Flop
DSP	Digital Signal Processing
MACC	Multiply and Accumulate

GTSRB	German Traffic-Sign Recognition Benchmark
SVM	Support Vector Machine
HOG	Histogram of Oriented Gradients
ReLU	Rectified Linear Unit
FSM	Finite State Machine
CLAHE	Contrast Limited Adaptive Histogram Enhancement
FI	Fault Injection
SEU	Single-Event-Upset
SBU	Single-Bit-Upset
MBU	Multibit-Upset
FPGA	Field Programmable Gate Array
RISC	Reduced Instruction Set Computer
CLB	Configurable Logic Block
CPU	Central Processing Unit
GPU	Graphics Processing Unit
TCAD	Technology Computer Aided Design
I2C	Inter-Integrated Circuit
CAN	Controller Area Network
AMBA	Advanced Microcontroller Bus Architecture
SMC	Static Memory Controller
ASIC	Application Specific Integrated Circuit
Caffe	Convolutional Architecture for Fast Feature Embedding
VLSI	Very-Large-Scale Integration
VHSIC	Very High Speed Integrated Circuits
VHDL	VHSIC Hardware Description Language
LMDB	Lightning Memory-Mapped Database

BSD	Berkeley Software Distribution
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
SPI	Serial Peripheral Interface

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Humankind has trusted on technology to get better living standards since the very early stages of the civilization. The construction of equipment and artifacts is always limited by the available technology. Starting with a stick as defense accessory and a lever to multiply its force, humankind learned to transform energy and to build complex mechanisms during the industrial revolution (DERRY; WILLIAMS, 1960). But it was the use of electricity as a support for signal processing and communication that enabled more sophisticated apparatuses for the daily life and process automation. Even though some digital operations were possible with pneumatic or electromagnetic relay circuits, it was the advent of digital electronic computer that pushed further the possibility of information processing and systems programmability (RABAEY; CHANDRAKASAN; NIKOLIC, 2002). But it is the evolution of the micro and nanoelectronics technology that is nowadays bringing to reality the dream of the electronic brain of the middle of the 19th century (HAYKIN; NETWORK, 2004).

Artificial Intelligence (AI) is an object of research in several domains and even commercial applications are running based on AI. Many tasks that are typically considered "human tasks" like language translation, face recognition, speech recognition and natural language processing are being automated. The most common architecture used in AI is the Neural Network (NN) that was traditionally built with a few levels, typically three, and is now using more than ten levels and is called "deep learning" or Convolutional NN (CNN) (GOODFELLOW; BENGIO; COURVILLE, 2016) (HAYKIN; NETWORK, 2004). The possibility of training such systems to control complex activities like to drive in a city traffic allowed the development of Advanced Driver Assistance System (ADAS) with the goal of decreasing the number of accidents. Also, this technology spurred the development of autonomous cars (XILINX-AUTOMOTIVE, 2016)(HUANG, 2016).

In the context of autonomous cars and ADAS, deep learning is used to emulate the vision of a driver (GOODFELLOW; BENGIO; COURVILLE, 2016). One of the visual tasks that can be performed by means of deep learning algorithms is traffic-sign recognition. In the International Joint Conference on Neural Networks (IJCNN) many machine learning algorithms were evaluated on GTSRB dataset and the Convolutional Neural Networks achieved excellent results even better than the individual human and average human performance (STALLKAMP et al., 2011). Therefore, using Convolutional Neural Networks is an excellent approach to classify traffic-signs.

Deep learning vision systems deployed in autonomous cars must have a fast response time. In addition, this system cannot consume high power to avoid high thermal dissipation. Graphics Processing Units (GPUs) achieve excellent performance results due to their thread-level parallelism but they have high power consumption (HUANG, 2016)(NVIDIA, 2016). Application Specific Integrated Circuits (ASICs) can achieve excellent performance with low power cost (LEE et al., 2017) but deep learning algorithms are constantly evolving, thus a platform with more flexibility has to be chosen. All-Programmable System-on-Chip (APSoC) alternatives are flexible, that is, it is possible to update the implemented deep learning algorithm. In general, the implementations in this platform obtain balanced power and performance results (XILINX-AUTOMOTIVE, 2016), for this reason, it is an excellent choice for autonomous cars and ADAS applications. However, the PL part of the APSoC platform has peculiar radiation effects, therefore these effects must be analyzed in order to characterize an implemented design under soft-errors (KASTENSMIDT; CARRO; REIS, 2006).

CNN training is a very time-consuming task, so the use of a framework is necessary (JIA et al., 2014). Therefore, the Caffe framework was used to train a case study CNN to classify 12 sub-classes within the super class "Prohibitory" from the GTSRB dataset. Initially, a CNN model was trained achieving an average accuracy of 90.0%. However, the implementation of this CNN model on the APSoC platform exceeded the available resources of the PL part. Then, the CNN model was reduced to one that classifies half of the prohibitory sub-classes. The CNN was re-trained and the accuracy results were preserved. A VHDL automatic generator for CNNs was developed in order to reduce the required design time for implementing CNNs in APSoCs.

The proposed CNN reliability was analyzed by means of fault injection by emulation. Three types of effects were considered in the reliability evaluation, errors, failures and un-Application Correct Execution (un-ACE). Errors are discrepancies in the CNN output which do not change the correct traffic-sign classification. Failures can be classified into critical SDCs and time-outs. Critical SDCs are discrepancies in the CNN output which change the correct traffic-sign classification, while time-outs mean the CNN does not send the response in the defined execution time. The un-ACE effect takes place when none of these effects occur (MUKHERJEE et al., 2003).

1.1 Motivation

Statistics show that the majority of accidents is caused by reckless driving (NHTSA, 2013), thus some accidents can be avoided if an autonomous car is available (GERLA et al., 2014). In addition, ADAS is also useful to reduce the chance of accidents happening (GERONIMO et al., 2010). Since driving is mainly performed using human visual senses, computer vision algorithms must be used to emulate human vision. Deep learning is a branch of AI that has many successful applications in computer vision, speech recognition, natural language processing and other areas of commercial interest (GOODFELLOW; BENGIO; COURVILLE, 2016).

Autonomous car applications are safety-critical and hard-real-time embedded systems. So, they must have a deterministic and fast response time in order to avoid accidents (BERGER, 2002) (LIU; NARAYANAN; BAI, 2000). If a deep learning vision system is deployed in an autonomous car, it must have a good and predictable execution time. Power consumption is also a concern in embedded system because it will lead to high thermal dissipation.

Electronic systems are susceptible to many sources of fault that can compromise safety of a computer vision application for an autonomous car. Therefore, the faults that can cause a failure or a time mismatch in the system must be analyzed and mitigated. They can be caused by manufacturing defects (stuck-at-0, stuck-at-1 and etc) (KAJIHARA et al., 1995), aging effects (WANG et al., 2007) or radiation effects (SEU, SET and so on) (BAUMANN, 2005). The International Organization for Standardization (ISO) established a standard for the automotive electronic production in 2011, the ISO 26262 (ISO-26262, 2011). This standard requires some fault modes to be analyzed during the design flow, these faults include stuck-at-0, stuck-at-1, Single Event Upset (SEU) and Single Event Transient (SET). On that account, a fault analysis methodology must be incorporated in the design flow to validate the design in terms of dependability (otherwise known as reliability) (SINHA, 2011).

1.2 Goals and Contribution

The main contribution of this work is the analysis of reliability in a traffic-sign recognition CNN operating under soft error and implemented on an APSoC platform. The related work analyzes the performance and average accuracy of their traffic-sign

recognition approaches, but does not analyze their reliability under soft-errors (LEE et al., 2017)(HAN; ORUKLU, 2014)(CIREŞAN et al., 2012)(WANG et al., 2013)(LIU et al., 2016)(YANG et al., 2016) (KASSANI; TEOH, 2017). The goals of this work are as follows:

- To train a deep learning algorithm to recognize traffic-signs in order to obtain good average accuracy results in a short time.
- To perform a hardware acceleration of the trained deep learning algorithm, minimizing its power, area and design time.
- To analyze the reliability of the proposed deep learning approach under soft-errors

Chapter 2 will show a survey concerning the deep learning state-of-the art algorithms and the characteristics of the embedded platforms to accelerate them, as well as the Convolutional Neural Networks foundations. In Chapter 3, the terminology of the radiation effects in integrated circuits will be introduced, then the ground-level effects of the radiation and the specific radiation effects in Field Programmable Gate Arrays (FPGAs) will be addressed. In Chapter 4, the methodologies and tools used for the Convolutional Neural Network training as well as the training results will be discussed. Chapter 5 will explain how the VHDL automatic generator works, then the implementation results will be discussed. In Chapter 6, the reliability analysis alternatives will be compared, then the Fault injection setup will be explained. In Chapter 7, the reliability results will be discussed, as well as their possible causes. Lastly, in Chapter 8, the conclusions will be given followed by prospective future projects.

2 DEEP LEARNING ALGORITHMS FOR TRAFFIC-SIGN RECOGNITION

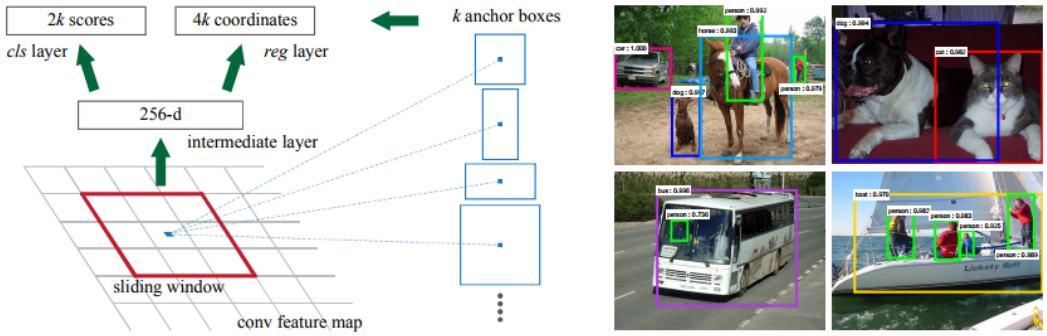
Autonomous cars are systems whose function is prompt delivery of the passengers to the destination with maximum safety and comfort and minimum impact on the environment (GERLA et al., 2014), whereas Advanced Driver Assistance System (ADAS) is an intelligent on-board system that aims to anticipate accidents in order to avoid them or mitigate their severity (GERONIMO et al., 2010). Some examples of autonomous car and ADAS tasks are lane detection, pedestrian detection, traffic-sign recognition and visual simultaneous localization and mapping (ALVES, 2017)(SANTOS, 2017)(HOELSCHER, 2017) (FUENTES-P; RUIZ-A; RENDÓN-M, 2015).

As autonomous car and ADAS tasks are emulations of the vision of a driver, they are implemented by computer vision algorithms. The computer vision algorithms, in turn, can be developed with computer vision libraries (BRADSKI; KAEHLER, 2008) or by deep learning frameworks (JIA et al., 2014). The difference between computer vision algorithms and deep learning algorithms is that in computer vision algorithms the programmer develops all the steps required for the computer vision solution, whereas deep learning algorithms use artificial intelligence algorithms to solve computer vision tasks. Artificial intelligence algorithms need a dataset to train the deep learning algorithm in order to learn some task (HAYKIN; NETWORK, 2004). For example, if one wants to recognize traffic-signs in an image, a dataset containing traffic-sign images samples must be presented to the deep learning algorithm in order to train it to perform a traffic-sign recognition.

Traffic-sign recognition activity is part of a more generic computer vision task called object classification. In order to recognize one object in an image, two steps are required: detection and classification. Detection is the process of scanning an image to find some objects. While classification is the process of placing these detected objects in classes (i.g. cars, pedestrians or traffic-signs), and then these objects are classified into sub-classes within their super classes (i.g different traffic-signs, types of cars) (YANG et al., 2016).

The state-of-the-art methods either use one machine learning algorithm to detect and classify objects or use image processing algorithms to detect objects and machine learning algorithms to classify these objects. The state-of-the-art machine learning algorithms that detect and classify objects are the Faster Region-based Convolutional Neural Network (Faster R-CNN) (REN et al., 2015) and the You Only Look Only-Once (YOLO)

Figure 2.1: Faster R-CNN Region Proposal Network (RPN)



Source: (REN et al., 2015)

(REDMON et al., 2016). The main contributions of the Faster R-CNN are described in the list bellow and Figure 2.1 shows the Faster R-CNN Region Proposal Network (RPN).

- Higher detection quality than R-CNN and Spatial Pyramid Pooling Network (SPP-net) (HE et al., 2014)(GIRSHICK et al., 2014)
- Training is single-stage, using a multi-task loss
- Training can update all network layers
- No disk storage is required for feature caching

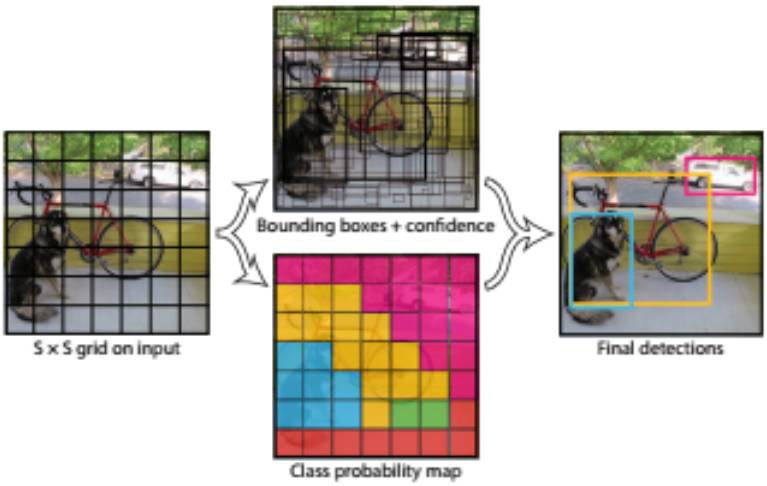
The YOLO CNN behavior is described in the list below and Figure 2.2 illustrates the object detection and classification.

- Divides the image into regions
- Predicts bounding boxes and probabilities for each region
- Bounding boxes are weighted by the predicted probabilities
- Threshold the detection by some value to only see high scoring detection

The scope of this chapter is only to evaluate the deep learning algorithms for traffic-sign classification, and the Faster R-CNN and YOLO are models that classify many types of classes. Therefore, the focus will be on deep learning algorithms that have distinct detection and classification blocks. Each block uses its own dataset, i.e. German Traffic-Sign Detection Benchmark (GTSDB) for detection and German Traffic-Sign Recognition Benchmark (GTSRB) for classification. The GTSRB dataset has 43 classes and more than 50,000 (39,209 for training and 12,630 for testing) sub-images and the GTSDB has 900 images (600 for training and 300 for evaluation) (STALLKAMP et al., 2011) (HOUBEN et al., 2013).

Wang (2013), employs a hierarchical methodology to detect and classify the German traffic-signs. The block diagram of his proposed method is illustrated in Figure

Figure 2.2: YOLO Detection System



Source: (REDMON et al., 2016)

Figure 2.3: Block diagram of the SVM method

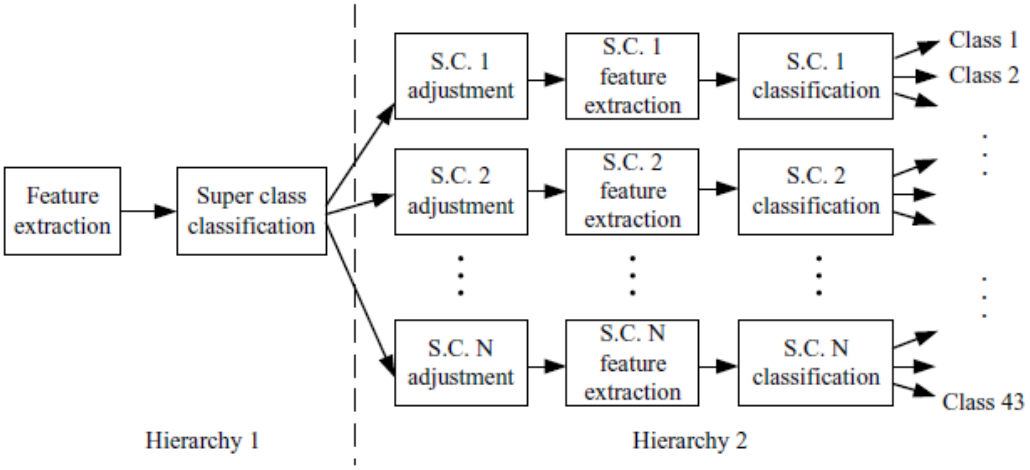


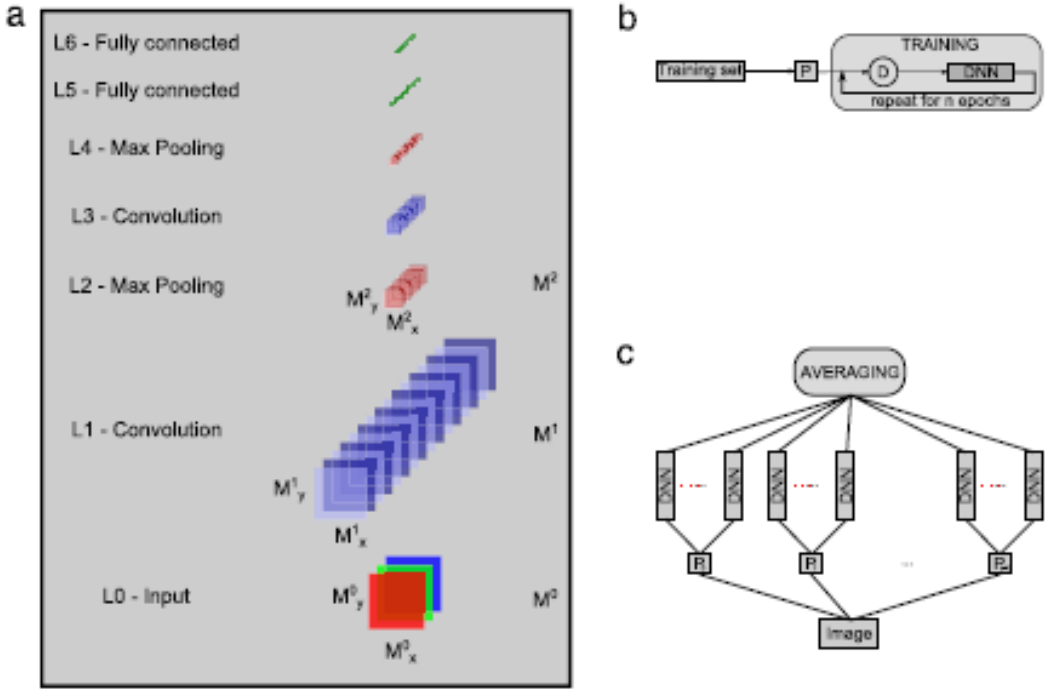
Fig. 3. Block diagram of the proposed method

Source: (WANG et al., 2013)

2.3. Firstly, in the detection stage, important features from traffic-signs such as colors and shapes are extracted using medium-level image processing algorithms. Then, these features are used to classify the proposals into super classes (Prohibitory, Danger, Mandatory, Derestriction and Unique) using a Support Vector Machine (SVM). Each super class has a different perspective adjustment. In the classification stage, the adjusted and detected traffic-signs are classified within super classes using another SVM, explaining the meaning of "hierarchical methodology". The key concept in (WANG et al., 2013) is the super-class-oriented adjustment, which improves and simplifies the classification process, reaching a state-of-the-art average accuracy of 99.52%.

Differently from the former work, the scope of the Multi-column Deep Neural

Figure 2.4: a) DNN architecture b) DNN trainig c) Multi-column deep DNN



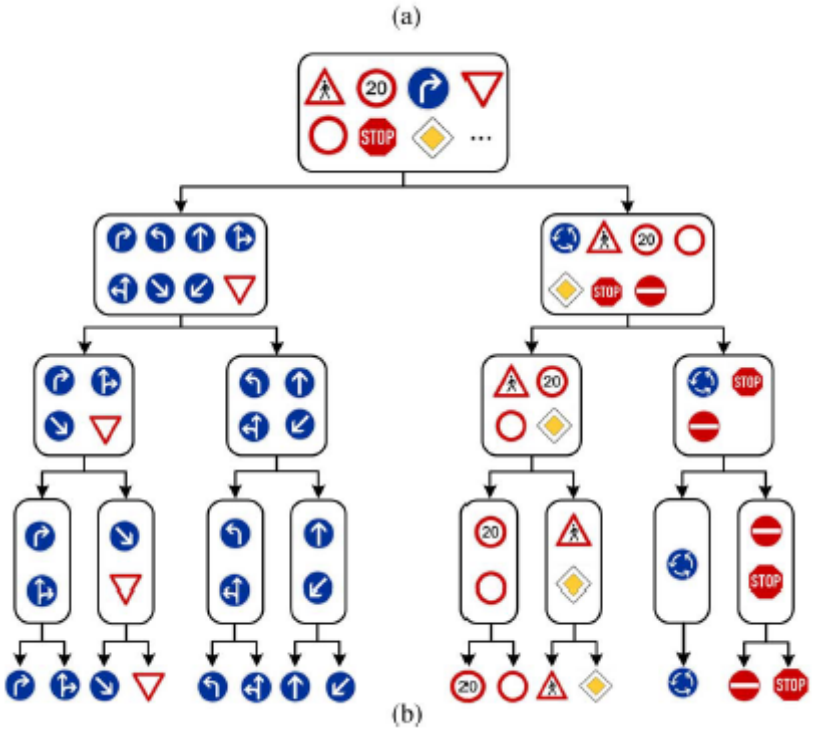
Source: (CIREŞAN et al., 2012)

Network (MC-DNN) (CIREŞAN et al., 2012) is only to classify the German traffic-signs, so only the GTSRB dataset is used. When using this approach neither super class nor sub-class is used, rather all 43 classes are classified by one complex DNN topology. "Multi-column deep" means there are 7 DNNs in parallel and the final output is the average of all of the DNN outputs. This methodology permits unseen features by one DNN to be seen by another one, which results in a high average accuracy of 99.46%. Figure 2.4 shows the a) Deep Neural Network topology, training process (b) and Multi-column DNN (c).

High Contrast Region Extraction (HCRE) and the Extended Sparse Representation (ESR) perform both the detection and the classification tasks (LIU et al., 2016). In the detection phase, the HCRE is performed based on the Viola-Jones-like detector and the Split Flow Cascade (SFC) Tree. Regarding the classifying phase, the ESR approach is based on an under-sampled face recognition algorithm which is considered to be an occlusion robust method. To such degree, this work achieves competitive accuracy (98.5%) and performance results for high resolution images. Figure 2.5 illustrates the detection process and Figure 2.6 illustrates the classification process.

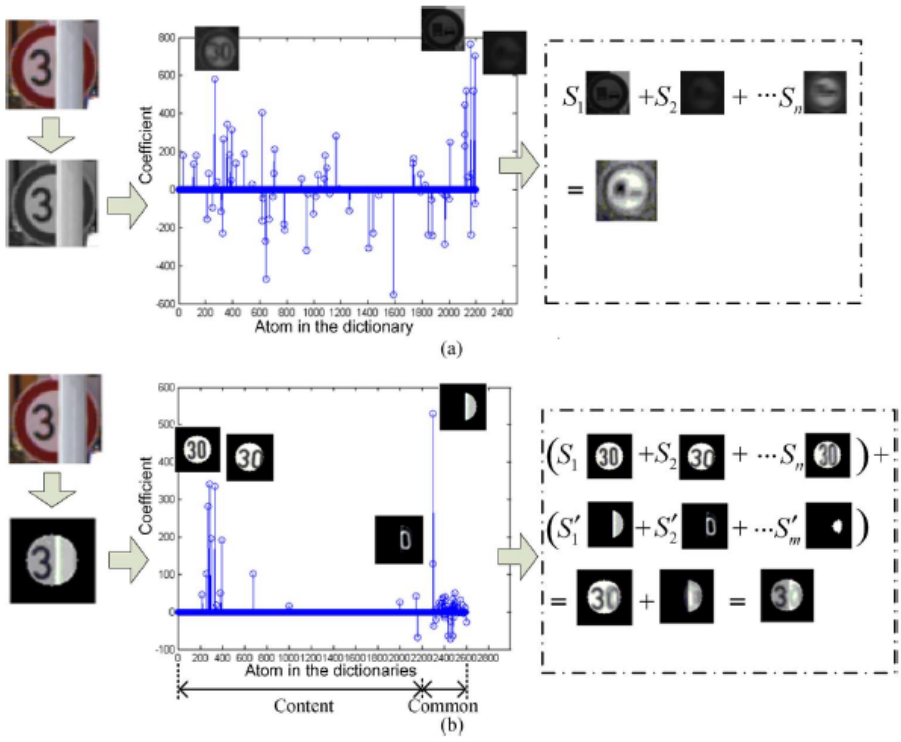
Soft Histogram Oriented Gradient (SHOG) (KASSANI; TEOH, 2017) is focused on the classification block. It is a variant of Histogram Oriented Gradient (HOG) approach because it exploits the symmetry shape of traffic-signs. Through this medium, SHOG is more discriminative than HOG. After the features are generated using the SHOG

Figure 2.5: a) Multi-block Normalization Local Binary Pattern (MN-LBP) features with threshold b) Split-Flow Cascade structure



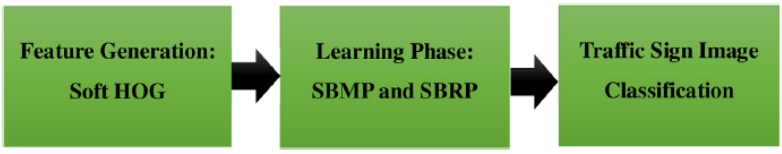
Source: (LIU et al., 2016)

Figure 2.6: a) Classical Sparse Representation Classification (SRC) method b) Proposed Extended Sparse Representation Classification method (ESRC) method



Source: (LIU et al., 2016)

Figure 2.7: Block diagram of the SHOG + SBRP/SBMP classification process



Source: (KASSANI; TEOH, 2017)

Figure 2.8: The pipeline of the traffic-sign recognition system

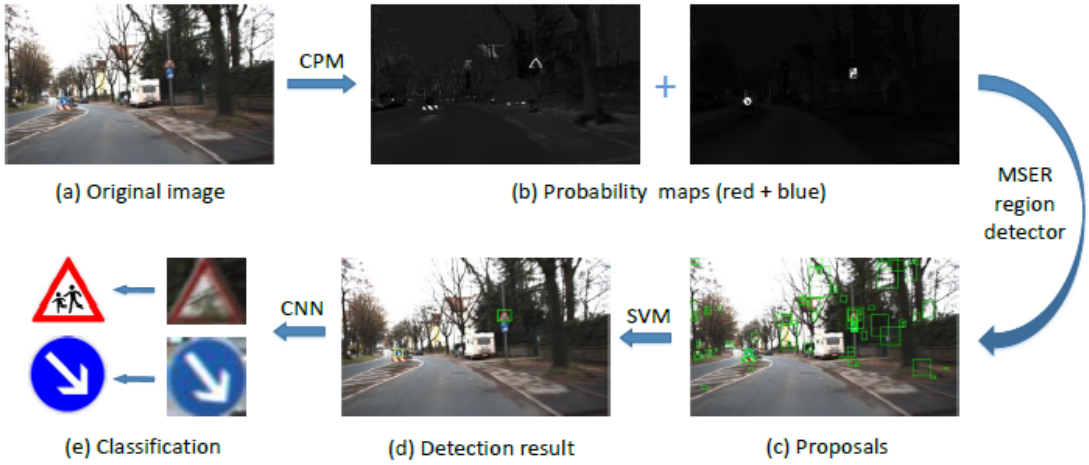


Fig. 1. The pipeline of the proposed traffic sign recognition system.

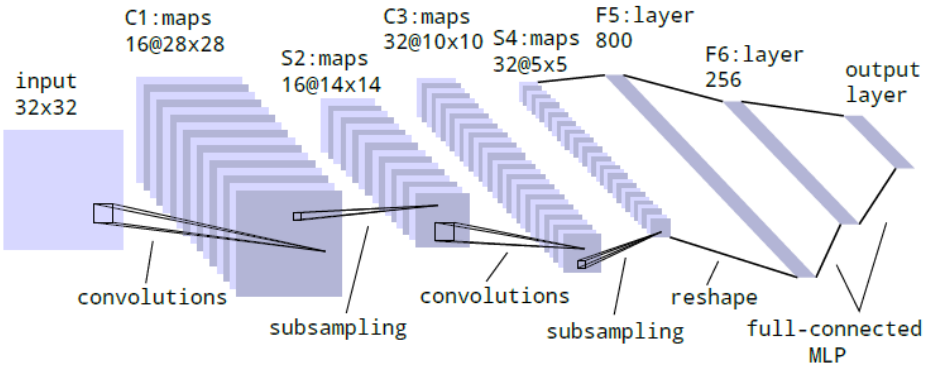
Source: (YANG et al., 2016)

approach, two training methods were introduced, the Sparse Bayesian Multivariate Polynomial (SBMP) and the Sparse Bayesian Reduced Polynomial (SBRP) model. These training methods enabled implicit feature selection and lead to a competitive average accuracy of 98.17%. Figure 2.7 shows the block diagrams involved in the classification process.

In the work of (YANG et al., 2016), the German traffic-sign detection is performed in four stages as seen in figure 2.8. Firstly, the images are processed by a Color Probability Model (CPM) that produces the probability of a pixel color belonging to a traffic-sign color (Figure 2.8 (b)). Then, the proposals are extracted using the Maximally Stable Extremely Regions (MSER) (Figure 2.8 (c)). After which, the proposal features are extracted using the HOG, and then are classified into super classes (Prohibitory, Danger and Mandatory) using a SVM (Figure 2.8 (d)). Thereafter, in the classification stage, one CNN per super class classifies the detected proposals into sub-classes (Figure 2.8 (e)).

Figure 2.9 illustrates the CNN topology used in the classification phase. This is based on a basic CNN, the LeNET (LECUN; BENGIO et al., 1995), which enables an excellent inference computation time and a competitive average accuracy of 97.75%.

Figure 2.9: Lenet-based Convolutional Neural Network



Source: (YANG et al., 2016)

2.1 Convolutional Neural Networks

Convolutional Neural Networks differ from the other Neural Networks by processing data that have grid-like topology, such as time-series that can be thought as 1-dimension-grid taking samples at regular intervals and image data that can be thought as 2-dimension-grid pixels. Convolutional Neural Networks (CNNs) were designed to extract features and recognize shapes with a high degree of invariance to translation, scaling, skewing and other forms of distortion. A Neural Network is said to be convolutional when it is composed by at least one convolutional layer. Regarding the CNN training, it is performed by supervised or unsupervised learning algorithms (HAYKIN, 1994) (GOODFELLOW; BENGIO; COURVILLE, 2016) (JAIN; MAO; MOHIUDDIN, 1996) (LECUN; BENGIO et al., 1995). The mathematical definition of convolution used in the Convolutional Neural Networks differs from the definition used in engineering and pure mathematics, indeed the correct term for the operation used in Convolutional Neural Networks is cross-correlation. However, it was popularized as "convolution" in literature.

The mathematical representation of a 2-dimension-discrete convolution is given in Equation 2.1, where i and j are the dimensional indexes, K is the convolutional kernel, I is the input, $*$ is the convolution-operation symbol, and lastly m and n are the sum indexes. The mathematical representation of a 2-dimension-discrete cross-correlation is provided in Equation 2.2. As it can be noted, the only difference between the convolution and the cross-correlation is that in the cross-correlation neither kernel nor input is flipped. Flipping operation is not done in cross-correlation operation because this property is not useful for neural networks (GOODFELLOW; BENGIO; COURVILLE, 2016). When cross-correlation is referred in this work, this will mean convolution, following the

literature.

$$S(i,j) = (K * I)(i,j) = = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.1}$$

$$S(i,j) = (K * I)(i,j) = = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.2}$$

2.1.1 Convolutional Layer

The Convolutional Layer is responsible for extracting features from input images. The convolutional neuron receptive fields are sub-images from the previous layers whose dimensions are defined by the kernel size, thus local features are extracted. Each resultant image is stored in a feature map and all the convolutional neurons from a feature map share the weight matrix that has the same dimensions of the kernel. The weight sharing reduces the number of trainable free parameters, reducing the capacity of the learning machine, which in turn improves its generalization ability (HAYKIN; NETWORK, 2004) (LECUN; BENGIO et al., 1995). The mathematical definition of a convolution for a single image is given in Equation 2.3 (GONZALEZ; WOODS, 2002).

$$g(x,y) = [\sum_{s=-a}^{a} \sum_{t=-c}^{c} w(s,t)f(x+s, y+t)] + b \tag{2.3}$$

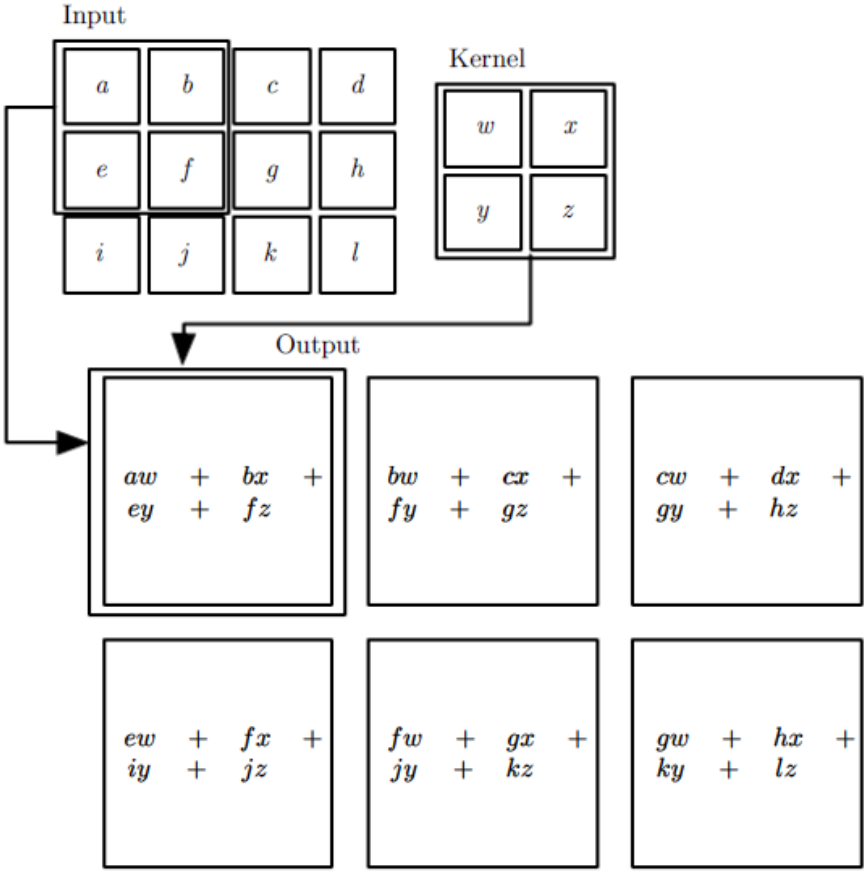
$$a = floor(kernel\_size\_x/2) \tag{2.4}$$

$$c = floor(kernel\_size\_y/2) \tag{2.5}$$

Where g is the resultant pixel from the convolution, x and y are the coordinates of the image, a and c are normalized sum limits calculated from the kernel sizes defined in Equations 2.4 and 2.5, w is the weight or kernel, f is the sub-image or window that is sampling the input image, and lastly b is the convolutional bias. An illustration of the convolutional computation is given in Figure 2.10. Note that when the kernel size dimensions are even values they cannot be normalized, because there is no central pixel.

Equation 2.3 described the mathematical operation of a convolutional layer that succeeds the input layer, i.e. there is only one input image. However, from the second

Figure 2.10: 2-D image Convolutional



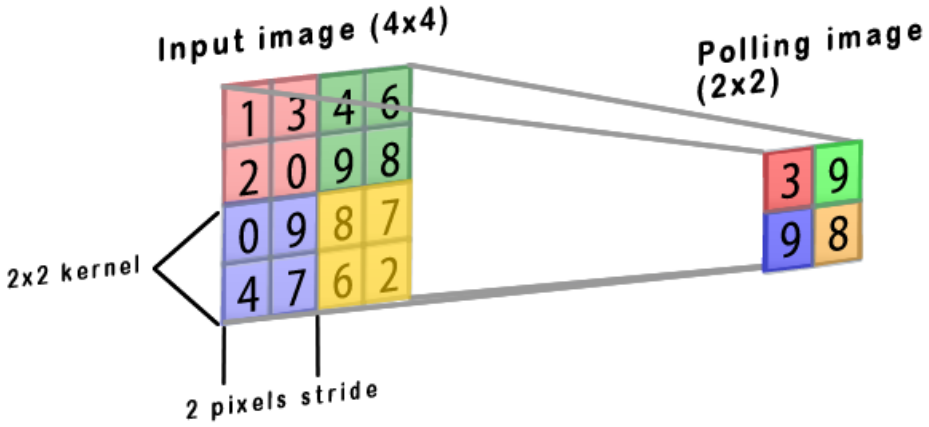
Source: (GOODFELLOW; BENGIO; COURVILLE, 2016)

layer onwards, the convolutional inputs are many feature map images instead of only one image, as such each convolutional neuron in the current layer computes the convolution for each one of the feature maps in the previous layer. Therefore, each one of the feature maps in the current layer has a weight matrix for each one of the feature maps in the previous layer. The mathematical representation of the convolution taking into account many input images are given in Equation 2.6 (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$g(x,y) = [\sum_{n} \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t,n) f(x+s, y+t, n)] + b \tag{2.6}$$

Where n is the feature map index of the previous layer which is used in the weight tensor and input images.

Figure 2.11: Max pooling illustration



Source: The Author

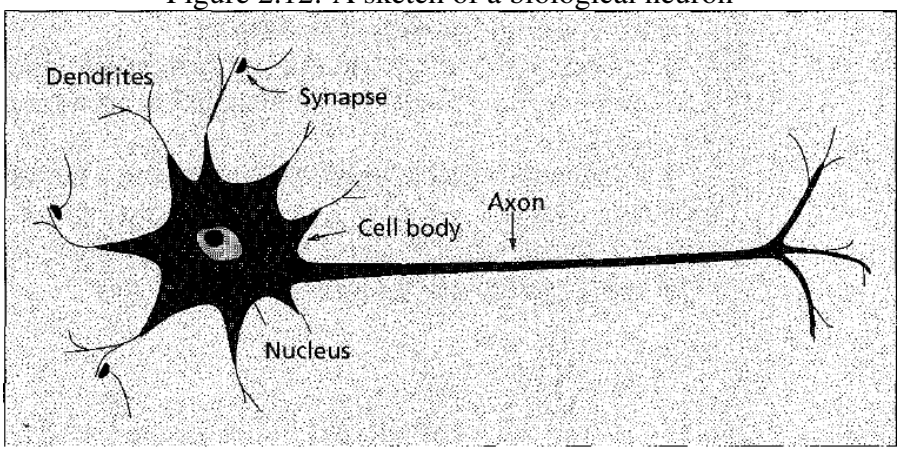
2.1.2 Pooling Layers

Each Convolutional layer is followed by a Pooling layer that performs the operation of subsampling, whereby the resolution of the feature map is reduced. The rate of the resolution reduction depends on the pooling kernel size and stride, as illustrated in Figure 2.11. This subsampling operation reduces the noise sensibility and other types of distortion. It is possible because pooling layers employ statistical ordering, which is also used in low-level image processing spatial filters for noise reduction (GONZALEZ; WOODS, 2002). The pooling layer can be employed using different functions, such as max, average and etc (HAYKIN, 1994) (GOODFELLOW; BENGIO; COURVILLE, 2016). The max pooling was used, therefore it is implemented in the hardware using max-comparator units .

2.1.3 Full-connected neurons

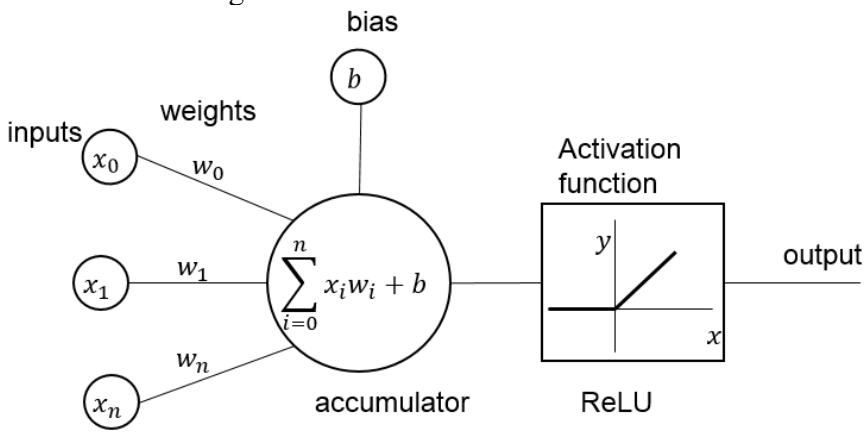
Full-connected (FC) neurons are Multi-Layer Perceptron (MLP) feedforward neural networks (HAYKIN, 1994). This class of Artificial Neural Networks (ANNs) is widely used in classification tasks and has many successful implementations in literature (HAYKIN; NETWORK, 2004) (GOODFELLOW; BENGIO; COURVILLE, 2016). The specialty of this neural network is pattern recognition, but can also be used in data analysis. The full-connected neurons are based on biological neurons, as illustrated in Figure 2.12.

Figure 2.12: A sketch of a biological neuron



Source: (JAIN; MAO; MOHIUDDIN, 1996)

Figure 2.13: Full-connected neuron



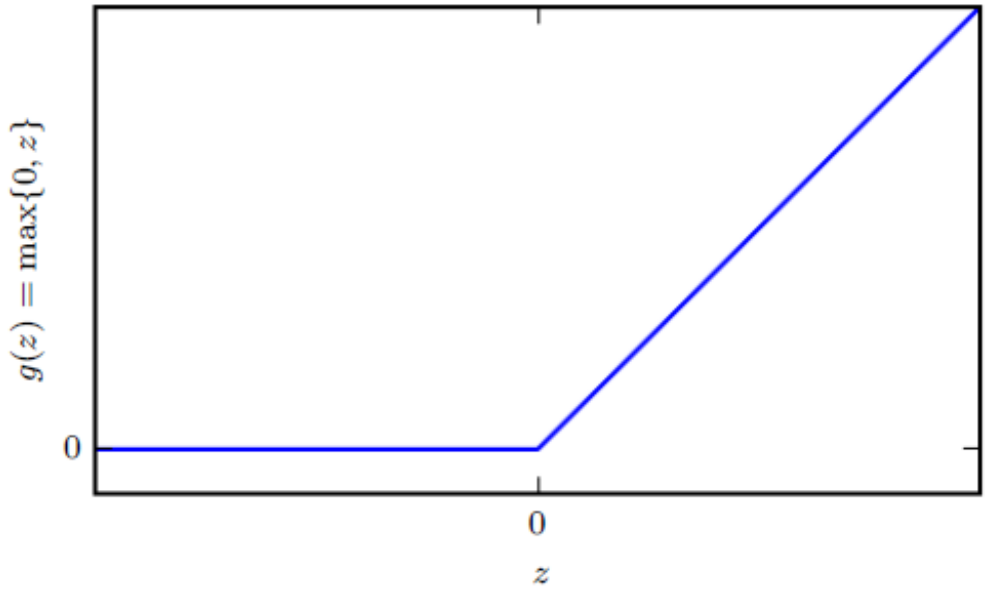
Source: The Author

So they have the dendrites, synapses, cell-body and axon. As illustrated in Figure 2.13, the dendrites are implemented by multiplying the inputs by the weights (synapses). The cell-body, otherwise known as "soma", accumulates the result of the multiplications and adds it with the bias. The axon performs the activation function (ReLU, which is described in the following) that is the output of the neuron and is connected to many other neurons (JAIN; MAO; MOHIUDDIN, 1996) (HAYKIN, 1994). In Hardware, the FC neuron computation can be performed either using MACC units or logic circuits.

2.1.4 ReLU Activation function

Rectified Linear Unit (ReLU) activation function employs the function $g(z) = max(0, z)$, as illustrated in Figure 2.14. It applies non-linearity either in Convolutional or FC neuron outputs but preserves some linear properties, simplifying the optimization process by learning algorithms (GOODFELLOW; BENGIO; COURVILLE, 2016). Due

Figure 2.14: ReLU activation function



Source: (GOODFELLOW; BENGIO; COURVILLE, 2016)

to its simplicity and efficiency, the ReLU is the default recommendation in modern neural networks (NAIR; HINTON, 2010) (GOODFELLOW; BENGIO; COURVILLE, 2016). In hardware, it has a considerable area-saving in comparison to sigmoid activation functions. The ReLU hardware implementation requires just a greater than 0 comparator.

2.1.5 Softmax function

The Softmax function is a way of improving the classification results. It computes the probability vector of the classes and increases the difference between the neuron outputs, as defined in Equation 2.7 (BRIDLE, 1990).

$$O_j = \frac{\exp I_j}{\sum_k \exp I_k} \tag{2.7}$$

Where O is the softmax output, I is the softmax input (FC neuron outputs), j is the index of the neuron being computed and k is the index of the other neurons. Softmax computation requires a great amount of hardware resources due to the sums of exponential, so it will be developed in software.

Figure 2.15: Nvidia DRIVE PX 2 technical characteristics

	TITAN X	DRIVE PX 2
Process	28nm	16nm FinFET
CPU	–	12 CPU cores 8-core A57 + 4-core Denver
GPU	Maxwell	Pascal
TFLOPS	7	8
DL TOPS	7	24

Source: (HUANG, 2016)

2.2 Acceleration platforms for Deep Learning

2.2.1 Embedded GP-GPUs

Nvidia corporation releases the first Graphics Processing Unit (GPU) for Personal Computers (PCs) in 1999, revolutionizing the PC industry (NVIDIA, 2017). Initially, this platform was intended to accelerate game applications, but due to its high computation power, it is also used in general-purpose applications, such as deep learning, medicine and etc, as such it is called General Purpose GPU (GP-GPU) (VISWANATHAN; HUSSEIN, 2017) (DING et al., 2011).

Deep-learning GP-GPU implementations achieve high performance results due to their massive parallel computation (NVIDIA, 2016). Thus, GP-GPUs are widely used in CNN training and CNN deployment (STRIGL; KOFLER; PODLIPNIG, 2010). GP-GPU had a high power consumption until last year. As such, it was an inappropriate approach to embedded system applications. Fortunately, in 2016, Nvidia releases the Drive PX 2, an embedded GPU that has a power consumption ranging from 10W to 250W (HUANG, 2016) (NVIDIA, 2016). The Power consumption depends on the operational mode where different chips from the board can be disabled. In addition to the GPU, it has 12 Central Processing Unit (CPU) cores and was manufactured in the 16nm (Fin-Field Effect Transistor) FinFET technology, as can be seen in Figure 2.15.

2.2.2 ASIC SoC and APSoC

When the Very-Large-Scale Integration (VLSI) design arises, the use of dedicated silicon wafers for a design (full-custom) was required, so this design is only economic viable for large quantity productions. One alternative to this style is the semi-custom design that comprises wafers with prefabricated logic gates distributed in array way (gate-array). The connections between the source and the sink of these gates are made by one metallization stage which is defined by the designer and made in a foundry, aiming to develop an Application Specific Integrated Circuit (ASIC) (SEDRA; SMITH, 1998).

Later, a new style of design emerged, the Field Programmable Gate Array (FPGA). This style enables the user to design VLSI systems without worrying about the high initial (non-recurrent) costs and the high design time. However, in general, it presents lower performance outcomes and higher power and area costs than an ASIC (SEDRA; SMITH, 1998). FPGAs use high density circuits in modern processes to design ICs that, as the name suggests, are completely programmable even after the product is sent, or programmable "in the field". There are two types of FPGA, the first is the one-time-configurable FPGA that uses a special process, such as fuse and anti-fuse to permanently program its interconnections and custom the circuit logic. The second type is the re-configurable FPGA that uses Static Read-Only Memories (SRAMs) or Flash memories to configure the routing and the logic functions, when SRAMs are used the configuration is volatile whereas when Flash memories are used the configuration is non-volatile (WESTE; HARRIS; BANERJEE, 2005).

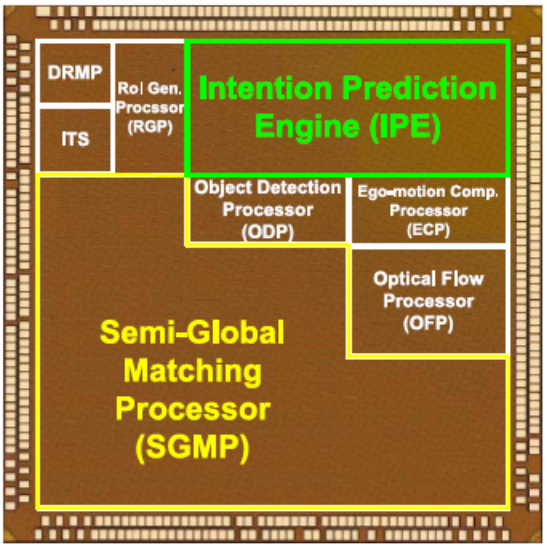
To deal with the trade-off between performance and power consumption, a figure of merit was created to evaluate implementations in hardware and select the best alternatives. This is the Power efficiency, as defined in Equation 2.8.

$$Power\_efficiency = \frac{GOPS}{Power\_consumption} \tag{2.8}$$

Where GOPS are Giga (Billions) Operations Per Second and Power Consumption is calculated in watts.

Lee (2017) implemented an ADAS ASIC System-on-Chip (SoC) that, as expected for an ASIC implementation, achieved state-of-the-art results of 560 GOPS with 30 FPS throughput under 720p stereo inputs and low power consumption (0.984 mW). Figure 2.16 illustrates the ADAS SoC micrograph whereby it is possible to see the area utilization of each block of the design.

Figure 2.16: ADAS SoC micrograph



Source: (LEE et al., 2017)

According to (DETTMERS, 2015)(GYSEL; MOTAMEDI; GHIASI, 2016), it is possible to convert the 32-bit floating point outputs of the accumulators of a deep learning algorithm into a 8-bit fixed point representation in order to reduce the area, power consumption but losing an insignificant average accuracy compared to the area and power saving. As can be seen in Table 2.1.

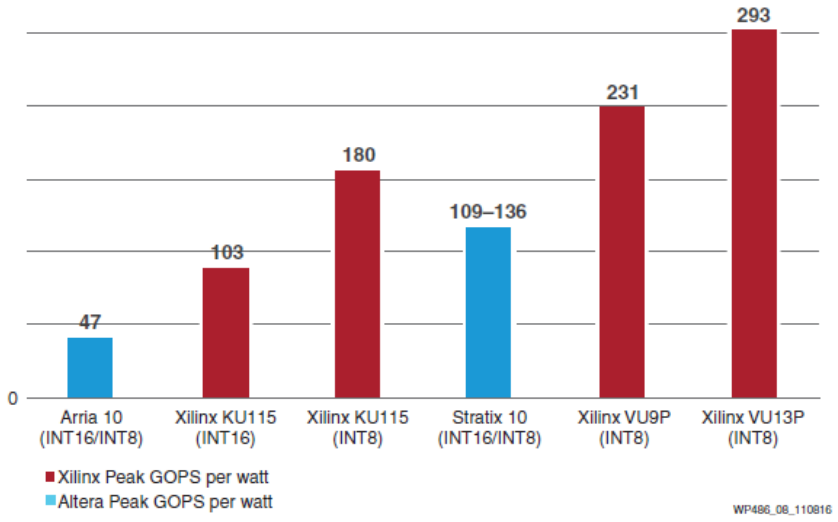
Table 2.1: CNN Models with Fixed-Point Precision. The numbers in brackets indicate accuracy without fine-tuning

	L. Outputs	CONV params	FC params	32-Bit FLP	FP Accuracy
LeNet (Exp1)	4-bit	4-bit	4-bit	99.1%	99.0% (98.7%)
LeNet (Exp2)	4-bit	2-bit	2-bit	99.1%	98.8% (98.0%)
Full CIFAR-10	8-bit	8-bit	8-bit	81.7%	81.1% (80.6%)
SqueezeNet top-1	8-bit	8-bit	8-bit	57.7%	57.1% (55.2%)
CaffeNet top-1	8-bit	8-bit	8-bit	56.9%	56.0% (55.8%)
GoogLeNet top-1 top-1	8-bit	8-bit	8-bit	68.9%	66.6% (66.1%)

Source: (GYSEL; MOTAMEDI; GHIASI, 2016)

Where "L. Outputs" are the Layer outputs, "CONV params" are the Convolutional parameters (convolutional weights), "FC params" are the Full-connected neuron parameters (FC weights), "32-bit FL" is the 32-bit Floating-point baseline, and lastly "FP Accuracy" is the Fixed-point average accuracy. For instance, in the first line of Table 2.1 the average accuracy of the LeNet (Exp1) using 32-bit floating-point representation was 99.1%, after the layer outputs, convolutional parameters and full-connected parameters are reduced to 4-bit, the average accuracy slightly decreases to 98.7% without fine-tuning (fine-tuning is the process of retraining the weights of a neural network to obtain a better

Figure 2.17: INT8 Deep Learning Power Efficiency Comparison: Xilinx vs. Intel



Source: (FU et al., 2017)

accuracy result). If fine-tuning is used, this value increases to 99.0% (GYSEL; MOTAMEDI; GHIASI, 2016).

When using this approach, it is possible to use 8-bit integer (INT8) optimization for deep learning implementations in either APSoC or FPGA (FU et al., 2017). Figure 2.17 shows excellent power efficiency results, in terms of GOPS / W, of deep learning architectures implemented on Ultra-scale Xilinx devices.

Han (2014) implemented both APSoC and FPGA trafic-sign recognition systems carrying out both the detection and the classification phases. The performances results of each implementation were compared where the Zynq-7000 APSoC implementation achieves a lower total execution time (96.5 ms) than the Virtex-5 FPGA one (777 ms), whereby 40x40 traffic-sign images were processed.

3 RADIATION EFFECTS IN INTEGRATED CIRCUITS

The development of high-density, functionality, and low-power circuits to meet the increasing consumer's demand for more powerful and capacity circuits, led a to dramatic increase in the radiation effects sensitivity of integrated circuits. Radiation effects in integrated circuits can occur in different ways, such as destructive effects, parametric shifts, data disruptions, and etc. Hard effects cause a damage in an integrated circuit while soft effects may cause an error or a failure in the device, but they do not cause a permanent change in the integrated circuit. Example of hard effect is the Total Ionizing Dose (TID) (OLDHAM; MCLEAN, 2003) which is predominant in space and military environments and it can be seen as a degradation of the component, like aging. Single Event Effects (SEEs), as the name suggests, are radiation effects induced by a single radiation event and are a major concern in space and in some commercial terrestrial applications. SEE can have hard and soft effects. SEE with soft effects, also known as Soft-errors, receives this name because the circuit is not permanently damaged by the radiation. Soft SEEs can be classified into Single Event Upset (SEU), Single Event Transient (SET) (BAUMANN, 2005) and Single Event Functional Interrupt (SEFI) (KOGA et al., 1997). SEE with hard effects are Single Event Burnout (SEB) (HOHL; GALLOWAY, 1987) and Single Event Latch-up (SEL) (STEPHEN et al., 1983), which are a major problem in space applications and must be tolerated by processor or hardening by design techniques.

SEUs occur when a radiation event changes the data state in a memory cell, latch or flip-flop. However, when a single radiation event has a very high energy, the deposited charge can be shared between adjacent bits or memory cells, consequently a Multi-bit Upset (MBU) can occur (MUSSEAU et al., 1996). SET is when a single radiation event occurs in the combinational logic of a circuit. The SET can propagate through the circuit and be latched in the sequential element (BAUMANN, 2005). However, SETs can be logically or electrically masked or even latch window masked, and in this way, they are not observed as errors in the sequential elements. SEFI is a single functional interrupt that occurs in the component, many time related to power issues, clock or configuration of the component (KOGA et al., 1997).

Regarding the physical circumstances, the magnitude of disturbance an ion crossing the semiconductor can cause depends on the Linear Energy Transfer (LET) which in turn depends on the particle energy, type of semiconductor and material mass. The collected charge Q_{coll} deposited by the ion depends on its type, trajectory and its energy over

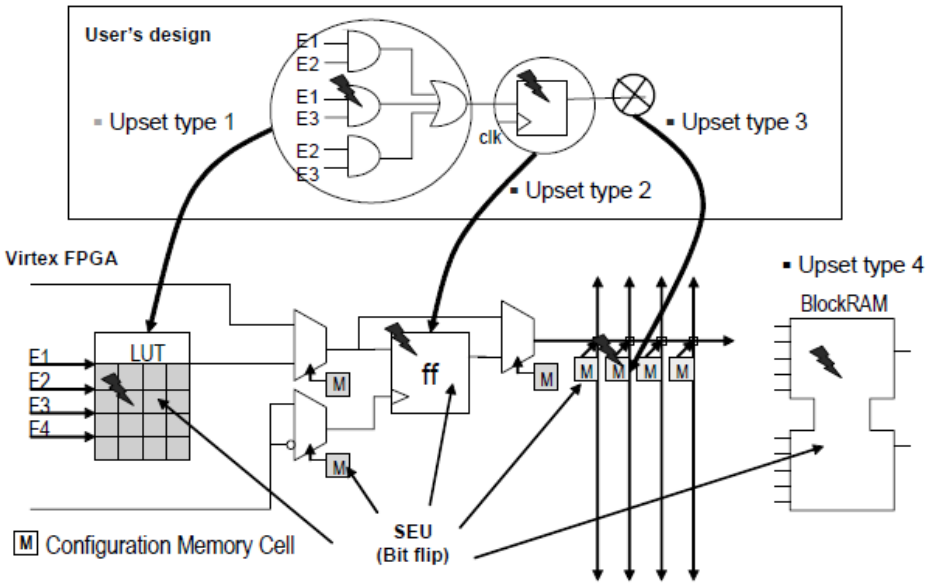
a path within or near the junction. The most charge-sensitivity parts of the circuit are the reverse-biased junctions. The amount of collected charge able to change a data state is the critical charge Q_{crit}. The critical charge is proportional to the node capacitance, operating voltage and strength of feed-back, and depends on the magnitude and temporal characteristics of the pulse generated by the collected charge. In DRAM memories the data value is changed when the collected charge exceeds the critical charge ($Q_{coll} > Q_{crit}$), whereas in SRAM memories beyond the collected charge condition ($Q_{coll} > Q_{crit}$) there is a factor that depends on the reaction speed of the circuit to the fault, i.e. slower reactions lead to an effective Q_{crit} increasing (BAUMANN, 2005).

There are three significant sources of ionizing particles in electronic devices in the terrestrial environment. The first source of soft-errors are the alpha particles which are emitted by the packaging materials. The Soft Error Rate (SER) from alpha particles can be reduced by purifying the packaging materials (BAUMANN, 2005).

The second source of ionizing particles in electronic devices is related to high-energy cosmic rays. Cosmic rays, collide on the atmosphere, producing secondary particles, such as protons, neutrons, muons and pions. However, less than 1% of the primary flux reaches the sea level and this flux is isotropic. At ground level, high-energy cosmic ray neutrons are one of the highest components of the particle flux and they are the most likely particle that can cause a device failure at ground level. Therefore, high-energy cosmic ray neutrons do not provoke directly the soft errors, they collide with the semiconductor material nucleus that in turn generates lighter particles which transfer energy to the circuit substrate. The high-energy cosmic ray neutrons are more likely to produce MBUs than alpha particles and low-energy cosmic ray neutrons. The device sensitivity to the effects of high-energy neutron collisions can be reduced by design or process modifications (BAUMANN, 2005).

The third source of ionizing particles in electronic devices is related to the low energy cosmic ray and thermal neutrons. Indeed, the third source is the secondary radiation induced from the interaction of low-energy cosmic ray and thermal neutrons, and boron atom. The source of neutrons from cosmic rays was explained previously. As boron is extensively used as a p-type dopant, low-energy cosmic ray neutrons has a significant effect in the boron-doped phosphosilicate glass (BPSG) dielectric layers. Advanced technologies that operate at low voltages are also susceptible to the effects of high energy cosmic ray neutrons interactions. The effects in BPSG dielectric layers can be reduced by changing the dielectric material or enriching the BPSG process without changing their

Figure 3.1: The comparison of the effects of a SEU in ASIC and FPGA architecture



Source: (KASTENSMIDT; CARRO; REIS, 2006)

desirable properties (BAUMANN et al., 1995).

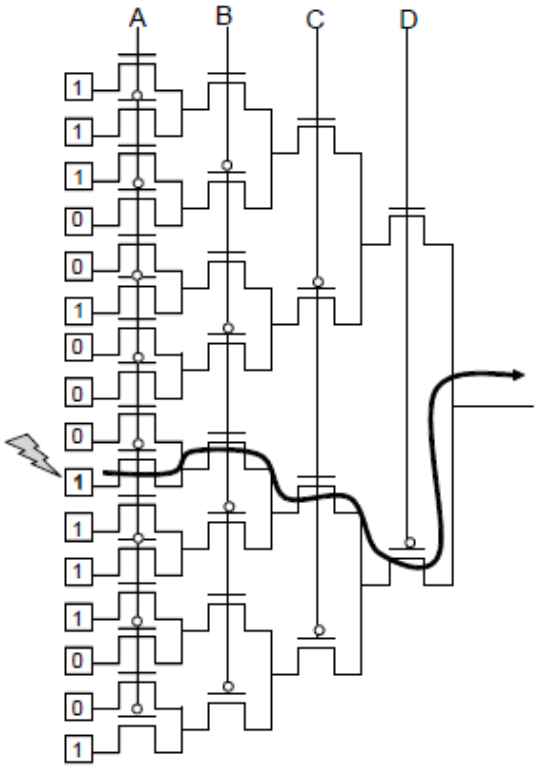
3.1 Radiation effects in SRAM-based FPGAs

There are peculiar radiation effects in SRAM-based FPGAs that do not occur in ASICs. In an ASIC, the effect of an energetic particle collision on the combinational circuit results in a transient fault. This transient fault is characterized as a transient logic pulse that may or not be latched in a storage cell, depending on the circuit delay and the topology. However, when a fault occurs in the sequential logic, the bit stored in this cell is inverted. An inversion in a logic state is called "bit flip" (KASTENSMIDT; CARRO; REIS, 2006).

In a SRAM-based FPGA, both the combinational and sequential logic are implemented by the user using SRAM configuration cells (volatile), as illustrated in Figure 3.1. Therefore, one fault in an ASIC combinational logic is equivalent to a bit flip in a FPGA Look Up Table (LUT) that performs the logic (Upset type 1) or in the cells that control internal routing by the multiplexers.

As the LUT performs the combinational logic (See Figure 3.2), one fault in a SRAM configuration cell will modify the implemented logic. Therefore, it will have a permanent effect unless a new configuration bitstream (File that contains the configurations bits to customize the logic) is loaded. Thus, an erroneous value of the combinational

Figure 3.2: Upset in the LUT (logic change)



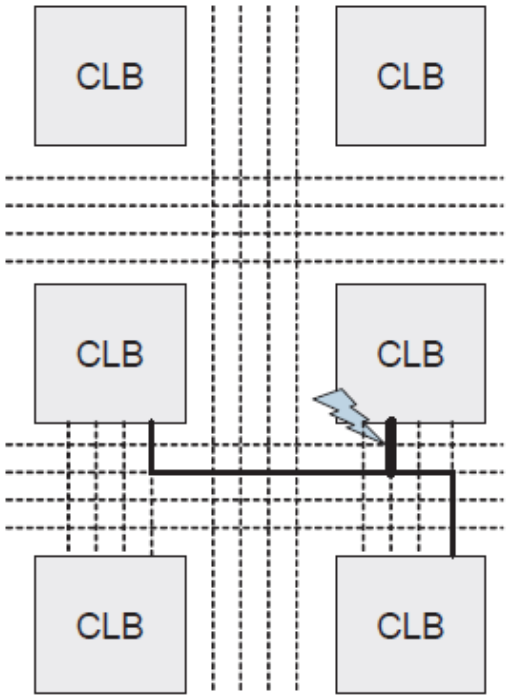
Source: (KASTENSMIDT; CARRO; REIS, 2006)

logic will be latched in the stored cell unless some fault detection technique is applied. However, faults that occur in the FPGA sequential logic (Figure 3.1 Upset type 2) will have a transient effect if the CLB flip-flop value is overridden by a new flip-flop load.

Other transient fault (SET) can occur in the multiplexer's inputs and outputs that control the CLB internal routing. SEU in the external routing (See 3.3) creates either "short" or "open" in the wires and has a permanent effect (Figure 3.1 Upset type 3) unless a new configuration bitstream is loaded. The Embedded memories (Block Random Access Memory - BRAM) are also susceptible to SEU and also have a permanent effect (Figure 3.1 Upset type 4). As the BRAM cannot be corrected by the bitstream without interrupting the normal operation of the application, it must be corrected by using fault tolerant techniques applied in the architectural or high-level description (KASTENSMIDT; CARRO; REIS, 2006).

Accordingly to (OHLSSON et al., 1998), as the transistor technology is shrinking, and consequently the logic density is increasing, the SRAM-based FPGA susceptibility to neutron induced soft-errors is also increasing. In a more recent work (NAZAR, 2013), this trend was confirmed in recent SRAM-based FPGA technologies. Tests were carried out by reading back the configuration bits of the FPGAs and verifying how many bit-

Figure 3.3: Upset in the routing (undesirable connection)

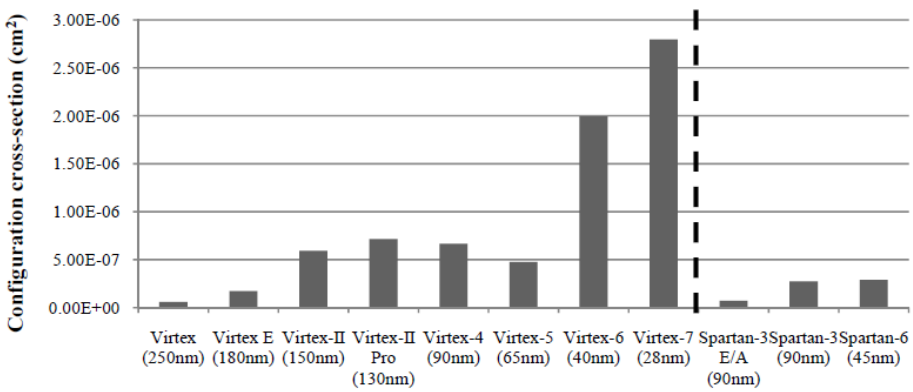


Source: (KASTENSMIDT; CARRO; REIS, 2006)

flips occur for a given fluence, i.e. static cross-section. It can be seen in Figure 3.4 that the static cross-section increases as the technology size decreases and the number of configuration bits available increases.

Considering these SRAM-based FPGA peculiar effects, a safety-critical system, as an autonomous car, must be characterized under soft errors in order to evaluate its reliability. Once the system is evaluated as vulnerable, some fault tolerant techniques must be applied.

Figure 3.4: Static cross-section for the configuration of the largest device of each family



Source: (NAZAR, 2013)

4 CONVOLUTIONAL NEURAL NETWORK TRAINING

4.1 Dataset

The German Traffic-Sign Recognition Benchmark was used as a training and test dataset. This is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) in 2011 (STALLKAMP et al., 2012). The goal of this dataset is to evaluate the performance of machine learning algorithms for German traffic-sign classification. This dataset contains 43 classes and more than 50,000 images in total, featuring an extensive and realistic dataset. In Figure 4.1 one sample of each class is shown. The dataset has many samples for each class, varying this position with respect to the observer, luminosity, background occlusion, and other types of variation in images. Figure 4.2 shows some variations on an image of one class.

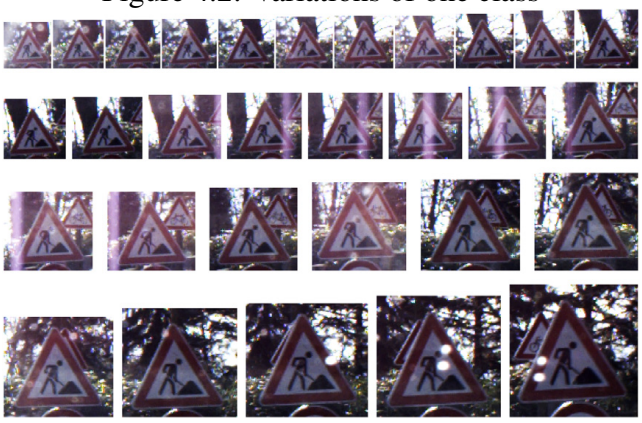
Aiming real-time applications, (YANG et al., 2016) concluded that it is more effective to develop one simple topology CNN for classifying the sub-classes within the super-classes, instead of developing one complex CNN to classify all the 43 classes. Thus, the 43 classes were divided into 4 super classes. Figure 4.3 show the (a) Prohibitory, (b) Danger and (c) Mandatory super classes.

The 8 remaining sub-classes belong to a super class referred to as "Others" that is separated from the other super classes, because they do not have common features. For instance, the super classes in Figure 4.3 have either circular or triangular shapes. The circular ones are either red or blue, and the triangular ones are only red. These super classes are easy of being separated by their color or shape features. Alternatively, the sub-classes within the super class "Others" have no common features, they have octagonal, triangular, diamond or circular shapes. Additionally, they are filled with red or yellow

Figure 4.1: One sample of each GTSRB class

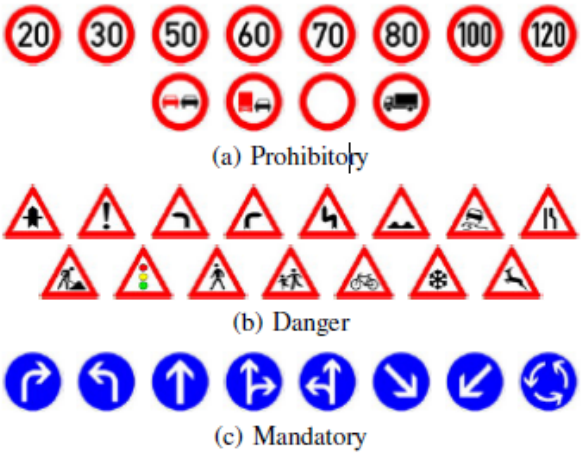
Source: (STALLKAMP et al., 2012)

Figure 4.2: Variations of one class



Source: (STALLKAMP et al., 2012)

Figure 4.3: Sub-classes of each super class



(a) Prohibitory

(b) Danger

(c) Mandatory

Source: (YANG et al., 2016)

Figure 4.4: One sample of each sub-class of the super class "Others"

Source: (STALLKAMP et al., 2012)

Figure 4.5: 6 classes of the Prohibitory subclass

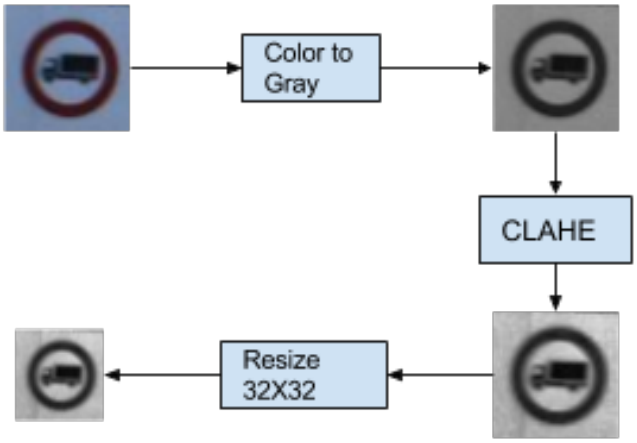
Source: The Author

colors or have a stripe or there is no sign in their center. Due to the fact that these classes do not have common features and cannot be clustered in super classes, they were grouped into a super class which is the result of an unexpected classification. For example, if one sub-image is not recognized as a "Prohibitory", "Danger" or "Mandatory" super class, it is recognized as an "Others" super class. Some "Others" super class samples are shown in Figure 4.4.

Due to resource limitations of our available SoC device, the CNN implementation was limited to a 6-class classifier, as such 6 sub-classes of the prohibitory super class were used. Thus, the dataset was reduced to 6 sub-classes, which comprises 9,930 images for training and 3,273 for testing. The 6 prohibitory sub-classes are shown in Figure 4.5.

As shown in Figure 4.2, the images were captured in different lighting and weather conditions, as such, images from the same sub-class present large differences and can be classified into different sub-classes. To deal with this kind of influence some pre-processing must be performed before the images are presented to the CNN classifier. The method used was the same of the work of (YANG et al., 2016) that employed a Contrast Limited Adaptive Histogram Equalization (CLAHE) (ZUIDERVELD, 1994) to adjust the contrast of the images. To apply the CLAHE to the images, the method CLAHE of the OpenCV library was used (BRADSKI; KAEHLER, 2008). As reported by (SERMANET; LECUN, 2011) the color plays little role in the classification, thus, following the work of (YANG et al., 2016), the original images were converted into grayscale to speed up the

Figure 4.6: Pre-processing steps



Source: The Author

CNN classifier. The same were converted into grayscale by using the OpenCV library (BRADSKI; KAEHLER, 2008). The images in the GTSRB dataset have different dimensions. Since the CNN input image dimensions are fixed, the images were resized. This pre-processing was performed using the Caffe framework, which will be described in the next section. All the steps that the pre-processing encompasses are illustrated in Figure 4.6. This pre-processing was performed during the design time, with an aim of improving the training accuracy. The real-time implementation of this algorithms is outside the scope of this work.

4.2 Framework for training

The model definition and training were performed with the Convolutional Architecture for Fast Feature Embedding (Caffe) framework. Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors (JIA et al., 2014). Yangqing Jia created the project during his PhD at University of California (UC), Berkeley. Caffe is released under the BSD 2-Clause license.

Some reasons for Caffe utilization are as follows:

- **Expressive architecture** - Models and optimization are defined by configuration without hard-coding. Switch between CPU and GPU by setting a single flag to train on a GPU machine then deploy to commodity clusters or mobile devices.
- **Modularity** - It has a catalogue within several state-of-the-art layers that can be

combined to develop your own model.

- **Speed** - Caffe can process over 60M images per day with a single Nvidia K40 GPU with the ILSVRC2012-winning SuperVision model and prefetching IO.

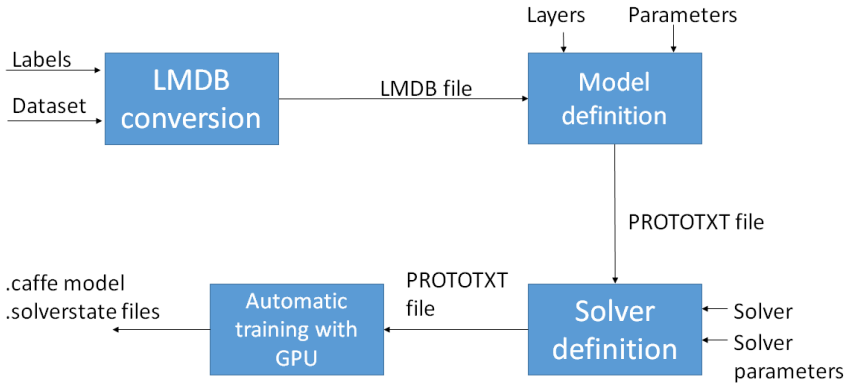
A Caffe model anatomy is composed of three components: layers, nets and blobs. Models are defined layer-by-layer where each layer performs a specific computation such as convolutional layer, max pooling layer, softmax layer and etc. The "blobs" are the memories of the layers where the framework stores, manipulates and communicates information such as forward signals and backward signals (derivatives). Lastly, the "net" is a collection of connected layers that characterize a specific model such as LeNet, Yolo, Faster R-CNN and so on. The "net" is defined from input data to "loss". Where "loss" is the error of the classification given an input data and its correspondent label. The layer inputs are defined as "bottom", whereas the layer outputs are defined as "top".

A Caffe model is trained using solvers which are parameterizable learning algorithms. The solvers optimize the model in two computational phases. The "forward pass" whereby the network outputs are computed and the backward pass where the gradients, which are used to optimize the network, are computed. (JAIN; MAO; MOHIUDDIN, 1996) (HAYKIN, 1994). The way by which the blobs are stored is managed by the Caffe framework. Therefore, the user will only need to define the model and solver parameters, the dataset, and the platform (CPU/GPU) to be used in training or deployment phase. The model and solver parameters are defined by a protocol buffer (.prototxt) and the learned parameters (weights) are serialized as binary protocol buffer (binaryproto) .caffemodel files using Google Protocol Buffer language. Protocol buffers enable efficient serialization, minimum-size binary strings (when serialized), interface implementation in multiples languages (manly C++ and Python) and other benefits. The implementation, communication and other issues are performed automatically by the framework.

4.2.1 Training process using Caffe

Before someone starts the training process with Caffe, the dataset must be prepared in order to be suitable for the framework. There are several ways of reading the dataset with Caffe, it can be read from the memory, operational system directory, data-base and so on. The Lightning Memory-mapped Database (LMDB) and batch processing improve the training performance being an excellent choice for big datasets. Caffe has an embedded

Figure 4.7: Steps for training a model with Caffe



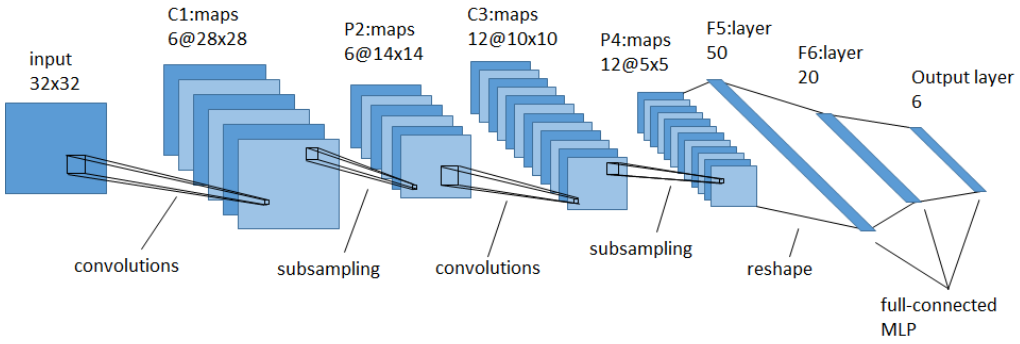
Source: The Author

code that converts an image directory and a .txt file (describing the image paths and labels) into a LMDB file. The GTSRB dataset contains a file describing the image directories and their labels but it is defined by a .csv file and contains other information. Thus, a Python script was created to adjust the .csv file to a .txt file suitable for the Caffe conversion code. Once the dataset is prepared for the framework, the model and solver parameters have to be defined in .prototxt files using the Google Protocol Buffer language. During the CNN training phase, the weights and the solver state are periodically saved into binary protocol buffer .caffemodel and .solverstate files, respectively, every 5000 iterations. These are the snapshots of the training and can be used to go back to a specific state of the training and use these snapshot trained weights for deploying and debugging. The use of the GPU is simply set by a flag in the Python code. The training steps with Caffe are summarized in Figure 4.7.

There is no specific method to determine which is the best Neural Network model for a given task. There is only heuristics to improve the success probability, so it must be discovered by the trial-and-error method (HAYKIN, 1994). However, well-known models in the literature can be used to accelerate the training process. Thus, the LeNet-based CNN model proposed by (YANG et al., 2016) was chosen due to its competitive accuracy and good performance results. However, as explained in section 4.1, this model is not suitable for the available APSoC platform, and then the CNN model had to be reduced to the model illustrated in Figure 4.8.

The reduced model parameters are shown in Table 4.1. Where the layers are shown line by line in a forward way and "in place" means the Full-connected neuron outputs are re-used by the ReLU activation function when the value is greater than 0 enabling memory saving. Data layers specifies how the images are read (back-end), some pre-processing operations, such as cropping, re-sizing, scaling and others, and how many

Figure 4.8: Reduced LeNet-based CNN model



Source: The Author

images are processed at time (batch size). Softmax with Loss layer is only used in the training phase while Accuracy layer is only used in the testing phase, both of them do not have parameters. Different batch sizes in the Data layer are used in the training and test phase. These model parameters were written in a prototxt file by using the Python interface with Caffe, simplifying the coding process.

Table 4.1: Parameters used in the model

Layers	Parameters
Data	batch size: 64 back-end: LMBD
Convolution 1	kernel size: 5 - feature maps: 6
Pooling MAX 1	kernel size: 2 - stride: 2
Convolution 2	kernel size: 5 - feature maps: 12
Pooling MAX 2	kernel size: 2 - stride: 2
Fully-connected 1	neurons: 50
ReLU 1	in place: true
Fully-connected 2	neurons: 20
ReLU 1	in place: true
Fully-connected 3	neurons: 6
Softmax with Loss	
Accuracy	

Source: The author

Once the model is defined, the solver type and solver parameters must be defined. Like the model definition, there is no specific method to define which are the solver type and solver parameters to a specific application, requiring the trial-and-error method (HAYKIN; NETWORK, 2004). Since the defined CNN model is based on the LeNET model (LECUN; BENGIO et al., 1995), the solver parameters used in a LeNET Caffe example (CAFFE, 2017) were also used. However, the solver type was defined by using the trial-and-error method. As it will be shown in the results to be seen ahead, the Adaptive gradient had the best accuracy result. This methodology reduces the training time because the Caffe framework has six solver types while the solver parameters combinations have

numerous possibilities. The solver parameters are shown in Table 4.2.

Table 4.2: Parameters used in the solvers

Parameter	Value
Learning rate	0.01
Momentum constant	0/0.9
Gamma	10E-5
Power	0.75
Test iterations	60
Test interval	500
Learning policy	inverse
Weight decay	5E-4
Max iterations 3	80000
Type	Adaptive Gradient

Source: The author

Where "Learning rate" is the rate by which the weights are updated, "Momentum constant" is the factor by which the previous weights are multiplied (Some solver types do not use momentum, thus its value is assigned to 0), "Test iterations" is the number of iteration will be done to generate the accuracy (Each one of the test iterations is done in all the images from the batch, whose quantity is defined by the batch size. So, for a 100 batch size, 6000 images are tested), "Test interval" is the number of training iterations until the point of a test execution, "Learning policy" is the way by which the "Learning rate" is decreased along the training time and is described by the equation 4.1, the "Weight decay", as the name suggests, is the factor by which the weights are decayed, "Max iterations" is the maximum number of training iterations, and lastly "Type" is the solver type.

$$learning\_rate = learning\_rate(1 + gamma * iteration)^{(-power)} \tag{4.1}$$

The Adaptive Gradient is described by the Equation 4.2.

$$(W_{t+1}) = (W_t)_i - \alpha \frac{(\nabla L(W_t))_i}{\sqrt{\sum_{t'=1}^{t} (\nabla L(W_{t'})_i^2)}} \tag{4.2}$$

Where W is the weight, t is the training iteration, i is the weight index of the CNN model, $t^{'}$ is the index that represents all the previous training iterations, α is the "Learning rate", and lastly L is the "Local gradient descent" (DUCHI; HAZAN; SINGER, 2011).

The solver parameters are also written in a prototxt file by using the Python inter-

face with Caffe.

4.2.2 Training results

The GPU used for the training was the Nvidia GTX 680 with 1500 CUDA cores and 2 Gigabyte Double Data Rate (DDR). Firstly, the solver accuracy results will be shown, then the training results using the best solver will be discussed, and lastly the false positives and false negatives. The training accuracy result for each tested solver is shown in Table 4.3. Regarding the solver parameters described in last section, the momentum constant was not used in some solvers, but the remaining solver parameters were used in all the solvers.

Table 4.3: Training accuracy for different solvers

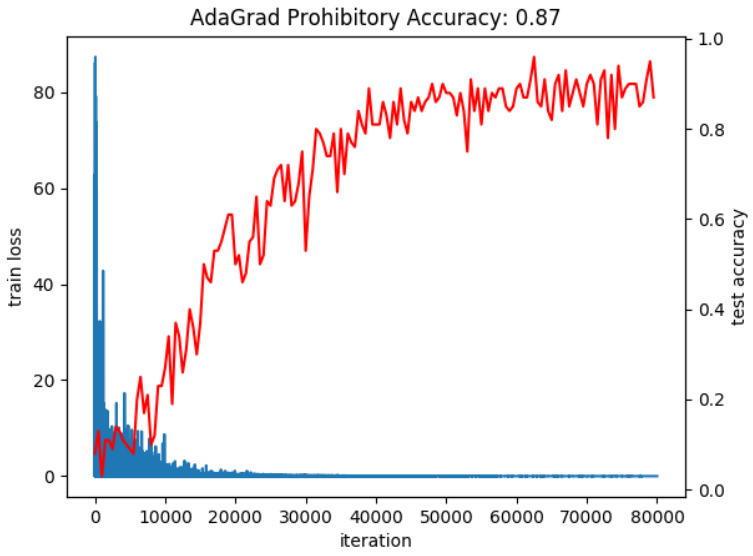
Solver	Accuracy
Adaptive Gradient	**90.1%**
Adam	14%
Nesterov Accelerated Gradient	7%
RMSProp	24%
Ada Delta	20%

Source: The author

As can be seen in the table, the Adaptive Gradient was by far the best solver for training 6 classes from the GTSRB dataset and using a LeNET-based CNN. This is due to the fact that the Adaptive Gradient solver takes into account the previous local gradients (DUCHI; HAZAN; SINGER, 2011) and the other solvers do not. Thus, if the training accuracy is decreasing due to a local minimum, the effect of this local gradient is decreased by the great previous gradients enabling that the weight space goes out from the local minimum, as defined by the Equation 4.2.

The plot of loss and accuracy in terms of the training iterations was generated in python by using the matplotlib library, as shown in Figure 4.9. Where the red line is the test accuracy and the blue line is the train loss. The accuracy is so oscillatory along the time. However, the accuracy tends to increase until a maximum value is reached. After which, the accuracy continues to oscillate about this maximum value. The oscillation of the accuracy can occur due to the dataset complexity and diversity. The CNN topology reduction did not change the accuracy, which proves that the CNN model reduction successfully fitted the classification capacity reduction. Although the final value is 87%, the maximum value is about 90% due to the oscillations. The training achieves acceptable

Figure 4.9: Training accuracy and loss

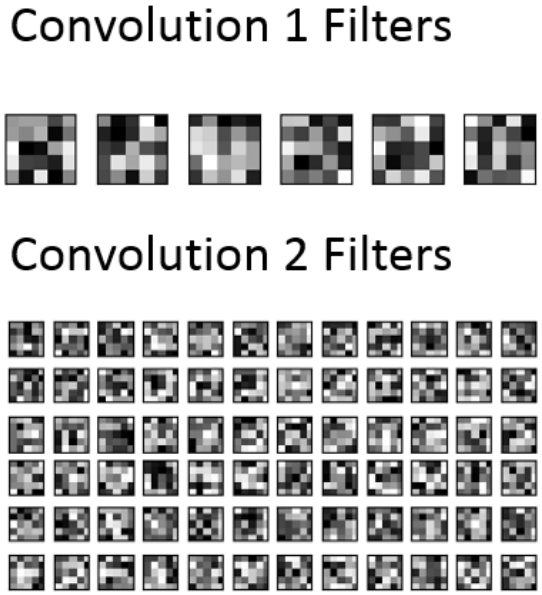


Source: The Author

results given that the solver parameters were not defined by exhaustive attempts. The initially proposed CNN model achieved an average accuracy of about 90.0%, where each super-class CNN training took about 10 minutes on a single GTX 680 GPU card, so the training of four super-class CNNs take 40 minutes. While the 43-class CNN training of (CIREŞAN et al., 2012), which achieves an average accuracy of 99.46%, took 37 hours on four GTX 580 GPUs cards. The 6-subclass CNN model training took only 2 minutes and the GPU usage enabled a speed-up in the training process of 40x in comparison with the CPU usage. Therefore, a more time-consuming approach could achieve state-of-the-art results.

The false positives and false negatives of each class are defined in Table 4.4. The most difficult sub-class of being recognized is the "sub-class 0" (20km/h speed limit). However, this sub-class has few samples (60) in comparison to other sub-classes that have four to seven hundred samples. Consequently, sub-class-0 accuracy does not reduce too much the average precision (89.88%). The easiest sub-class of being recognized is the "sub-class 2" (50km/h speed limit). However, it has many false-positives, which indicates that this sub-class is so easy of being detected so that other sub-classes can be confused with it. This is the trade-off between accuracy and false positives inherent in deep learning algorithms. If one increases the threshold of the classifier, the number of false negatives can be decreased, but the number of false positives can be increased. Alternatively, if one decreases the threshold of the classifier, the number of false positives can be decreased but there is a chance of the number of false negatives being increased. Nonetheless, it

Figure 4.10: Convolution filters



Source: The Author

also depends on the dataset. If "sub-class 0" had more samples like the other sub-classes, it could have a better accuracy result. In order to facilitate comparison with related work, the dataset test was not altered.

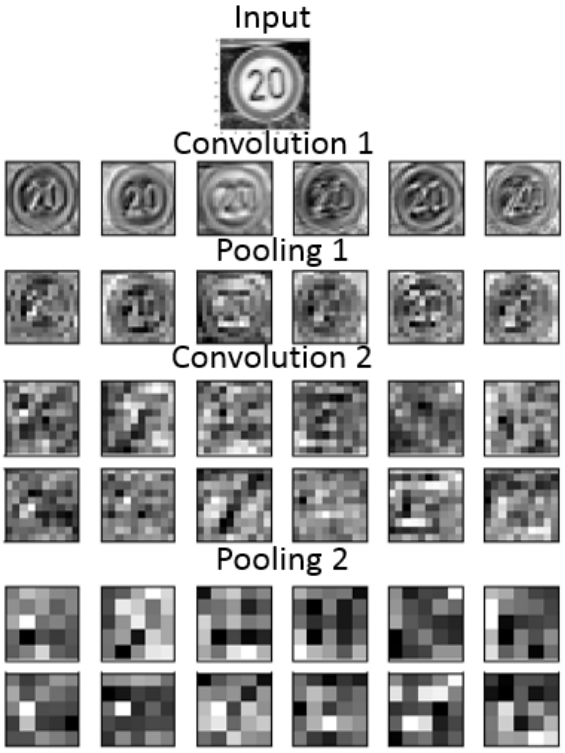
Table 4.4: False negatives and False positives

Sub-class	Samples	Accuracy	FN	FP
0	60	35.0%	38	20
1	720	91.25%	63	87
2	750	96.53%	26	97
3	450	87.77%	55	56
4	660	91.81%	54	43
5	630	80.63%	122	55

The Convolutional filters trained by the Caffe framework are shown in Figure 4.10 and the blobs of each layer for an input image from the sub-class 0 are shown in Figure 4.11.

The input image is filtered by the filters of Convolutional layer 1 (Figure 4.10) producing 6 feature maps, after which Max Pooling is computed subsampling the images, then all Pooling-1-feature-map images are filtered by the Convolutional-layer-2 filters (Figure 4.10) producing 12 feature maps. Lastly, the Convolutional-layer-2 feature maps are sub-sampled by the Pooling layer 2, producing the features of the input image that will be used to classify this image by the Full-connected neurons.

Figure 4.11: Feature images



Source: The Author

5 DEVELOPMENT OF CNN TOPOLOGY

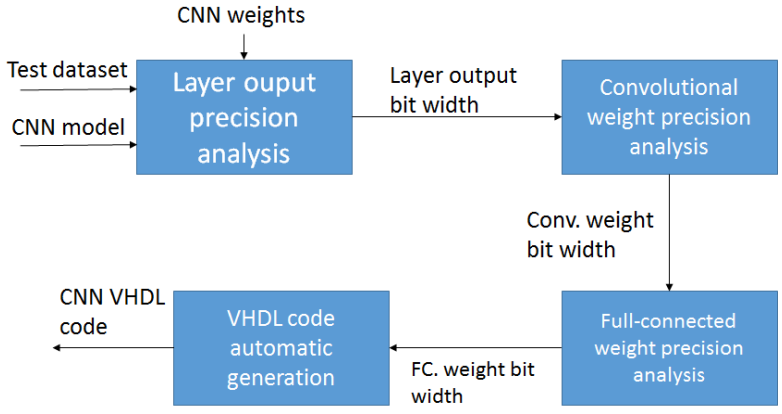
5.1 Automatic generator

Convolutional Neural Networks are massively parallel applications that comprise several layers and many feature map images or neurons within each layer. So the hardware description of a CNN is a difficult and repetitive task to be manually made. Therefore, a tool that automatically generates the VHDL of CNNs was developed. This tool uses Python interface with Caffe to access the internal signals and weights of a trained CNN and generates its VHDL code.

The proposed tool generates a timing-multiplexing architecture for LeNET based CNNs, but can also be extended to other CNN models and architectures. Before starting VHDL coding of an algorithm that uses floating-point representation, a precision analysis must be performed to determine the required bit width for signals. Thus, it is possible to reduce area and power, and improve performance by using fixed point representation. The signals that must be analyzed in CNNs are the layer outputs, the Convolutional parameters (weights) and the Full-connected parameters (weights). The Max Pooling layers and the ReLU activation function do not need to be analyzed because they do not perform arithmetical computations. However, if other pooling layers or activation functions were used, such as average and sigmoid respectively, a precision analysis would be required. The steps to automatically generate CNN VHDL codes are illustrated in Figure 5.1.

In order to reduce area utilization, fixed point and two's complement representation were used instead of floating-point representation. In a fixed-point representation the bit vector is divided into 3 parts: sign-bit, integer part, the fractional part. In order to save

Figure 5.1: CNN automatic generation steps



Source: The Author

area, the bit width of the integer part must be determined by Equation 5.1.

$$n = \begin{bmatrix} ceil(log_2(max\_value)), if max\_value > abs(min\_value) \\ ceil(log_2(abs(min\_value))), otherwise \end{bmatrix} \tag{5.1}$$

Where n is the number of the bit width required, max_value is the maximum value of a signal array in interest calculated over all the dataset, min_value is the minimum value of the same array and abs the absolute value. As the negative range in two's complement representation is 1-bit greater than the positive range, if the max_value is exactly equal to 2^n, the ceil function will not round the log result and it will lead to an overflow in the hardware implementation. Although it has a little probability, it must be taken into account. Equation 5.2 describes how this exception is treated.

$$n = \begin{bmatrix} ceil(log_2(max\_value)) + 1, if log_2(max\_value) - ceil(log_2(max\_value)) = 0 \\ ceil(log_2(max\_value)), otherwise \end{bmatrix} \tag{5.2}$$

The bit width of the fractional part can be determined by decreasing the fractional precision down to the average accuracy, calculated over all the dataset, be decreased. The last precision that does not change the accuracy will be the minimum precision. However, as mentioned in section 2.2, Xilinx FPGAs or APSoCs achieve good power efficiency outcomes when performing arithmetical operations using 8-bit integer signals at the cost of a slight accuracy loss (FU et al., 2017). All the trained weights by Caffe only have the fractional part out of the fixed-point representation (all the weights are < 0), whereas the layer outputs have all the parts. Therefore, following the Xilinx recommendation, the fractional bit width of the Convolution and Full-connected weights were truncated to 8 bits in order to achieve good power efficiency results. After performing the FC and Convolutional-layer-output precision analysis, it was discovered that decreasing the fractional part of the layer output signals do not have any effect in the average accuracy. Therefore, only the integer part out of the fixed-point presentation was used, which is defined by the Equations 5.1 and 5.2.

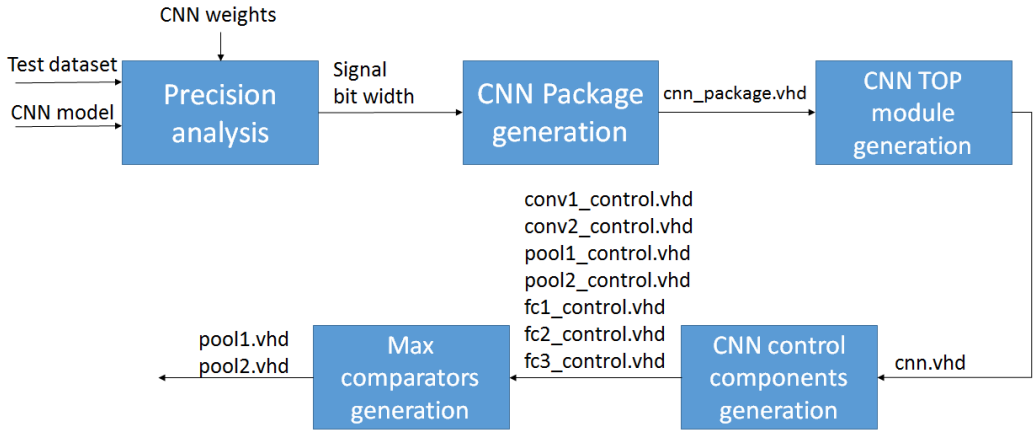
Even with the weight-precision reduction, the average accuracy only decreased from 89.88 to 88.88, which proves the (FU et al., 2017)(GYSEL; MOTAMEDI; GHIASI, 2016) conclusion. After the signal bit widths are determined, the CNN VHDL code was

Table 5.1: CNN signal bit widths

Signals	Bit Width
Convolution weights	9
Full-Connected weights	9
Convolution 1 outputs	10
Convolution 2 outputs	10
Full-connected 1 outputs	11
Full-connected 2 outputs	9
Full-connected 3 outputs	8

Source: The author

Figure 5.2: CNN VHDL code generation steps



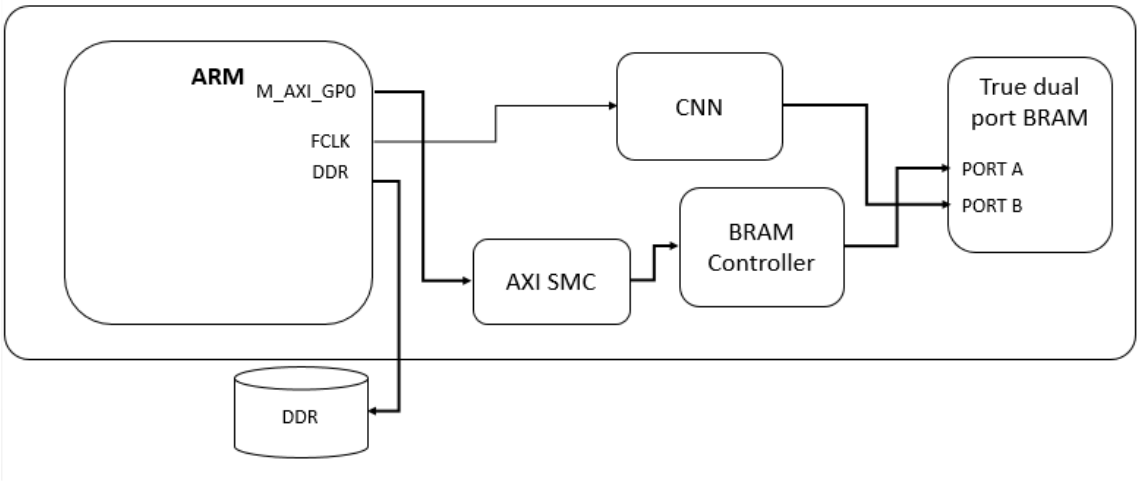
Source: The Author

generated using the steps defined in Figure 5.2. First, the signal bit widths are used to write the signal types, signal arrays and weight constants of the CNN to a VHDL package file. Then, the top module, which contains all the component declarations, component and MACRO instantiations, and signal assignments, is generated. The Convolutional and Full-connected components were not coded because they are instantiated as DSP MACC macros in the top module. After, the control components, which define the receptive field of each layer, are generated. Lastly, the max comparators of the Max-pooling layers are generated. The implementation details will be explained in the next section.

5.2 Timing Multiplexing Architecture

As described in sub-section 2.1.5, the softmax function is composed by sums of exponential, and as such, if it is implemented on the Programming Logic (PL) part of the APSoC it will expend a lot of FPGA resources. Therefore, it was developed in software running at the Processing System (PS) part of the APSoC, the ARM Reduced Instruction

Figure 5.3: CNN implementation in the APSoC platform



Source: The Author

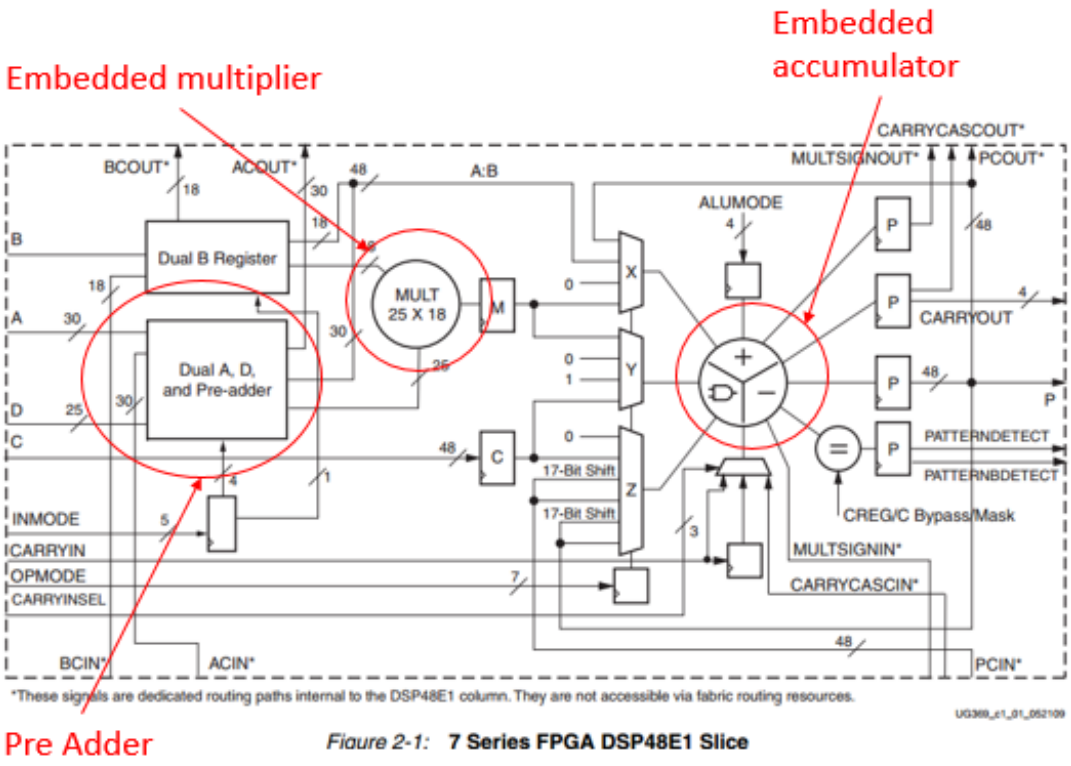
Set Computer (RISC) processor. The remaining functions of the CNN were implemented on the PL part of the APSoC. The whole system is illustrated in Figure 5.3.

The ARM processor communicates with the FPGA through a True dual-port BRAM by using an Advanced eXtensible Interface (AXI) BRAM controller. The dataset is stored in the DDR. Thus, the ARM processor reads an image from the DDR, after which, the same image is sent to the BRAM followed by a "start" signal. Then, the CNN performs the computations and, after a certain number of cycles, it writes the results and the "done" signal into the BRAM. After, the ARM processor reads the results, converts them into floating-point (due to the math.h library requirement) and performs the softmax computation.

All the components and macros are instantiated in the CNN top module, as mentioned in the last section. Convolutional and Full-connected computation are basically Multiply-And-Accumulate (MACC) operations. There are two ways of implementing MACCs in FPGA. They can be implemented using CLB resources of the FPGA (LUTs and CARRY logics) or by Digital Signal Processing (DSP) slices. According to (FU et al., 2017), DSP is a good choice for implementing deep learning algorithms, because it has dedicated routing and arithmetic resources by which good power efficiency outcomes can be achieved. Moreover, LUTs are generic resources, so it is recommended to leave these resources for the remaining logic of the CNN. The DSP slice is illustrated in Figure 5.4, as indicated, there are an embedded multiplier and an accumulator enabling MACC operation. Although, it has many signals to be customized they can be automatic assigned to constants using a MACRO MACRO in VHDL.

The max comparators used in the Pooling layers were implemented using high-

Figure 5.4: Xilinx 7 Series DSP48E1 Slice



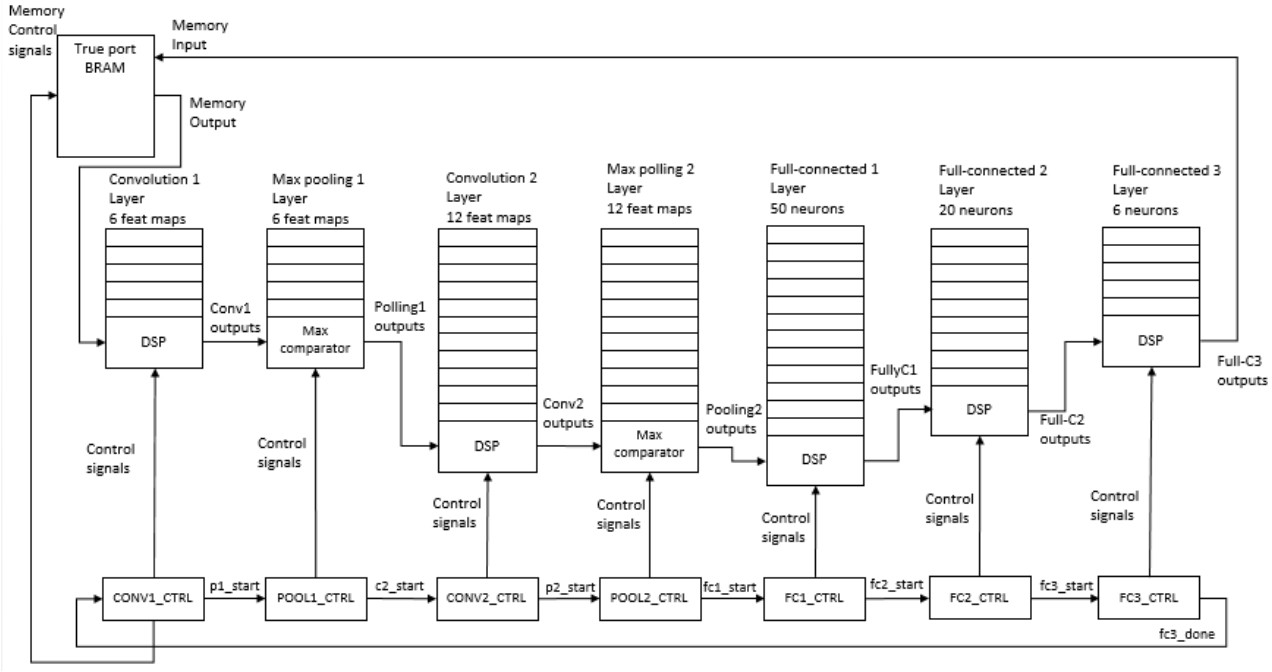
Adapted Source: (XILINX, 2016)

level VHDL and were synthesized to multiplexers, greater-or-equal comparators and "AND" gates. Once the fundamental components are created they are instantiated in the top module as illustrated in Figure 5.5.

As illustrated, the CONV1_CTRL (Convolution-1 control component) is responsible for controlling the true dual-port BRAM and the DSPs within the Convolutional layer 1. It receives the fc3_done signal that indicates the conclusion of the Full-connected-3 computation, then the results are stored into the BRAM. This is the most important control component in the network. There is one control component for each layer, whose function is to control the next layer receptive field (defined by the kernel sizes) and the current inputs and weights used as DSP operands (only in Convolutional and Full-connected layers). The CNN architecture is called "Timing multiplexing" because, in the DSP level, inputs and weights are multiplexed during the time, but analyzing at the top level the CNN architecture is defined as pipeline.

The simplified Finite State Machine (FSM) of the CONV1_CTRL component is shown in Figure 5.6. It has the generic "Reset" and "Idle" states, a state to read the "start" signal from the BRAM, a state to clear the offsets used to slide a window over the image stored in the BRAM, the state where the MACC operation is performed in the

Figure 5.5: CNN implementation top module



Source: The Author

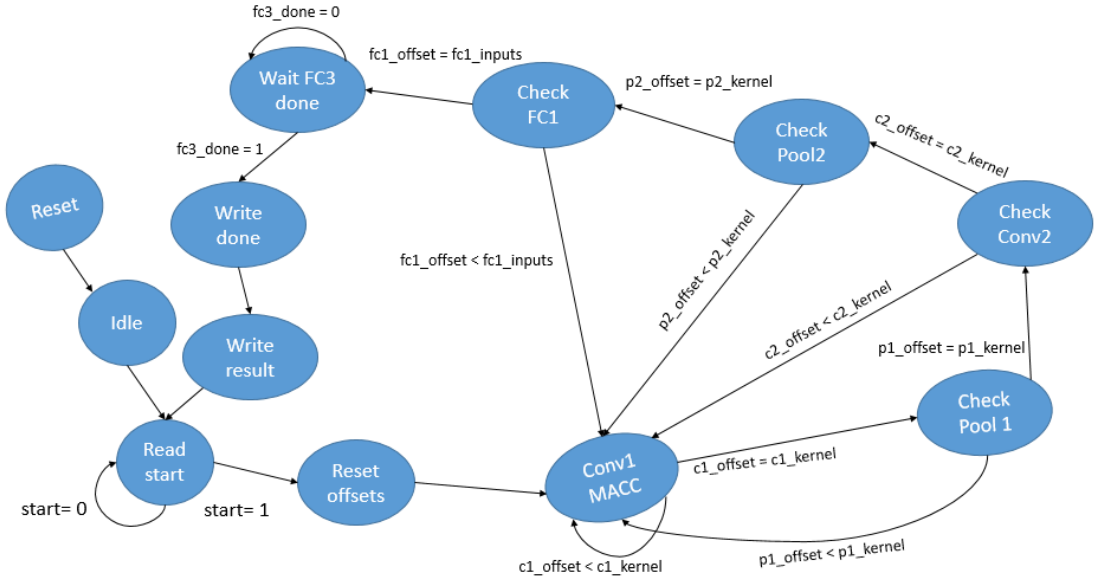
Convolutional layer 1, states for checking if an offset reached the kernel size of its layer, a state for waiting for the fc3_done signal and lastly a state to write the CNN results and "done" signal into the BRAM. For sliding a window over the input image stored in the BRAM, CONV1_CTRL is the most complex control component in the CNN.

The process of sliding a window in two dimensions over the input image stored in the BRAM (sweeping) was performed taking into account the next-layer receptive fields, i.e. their kernel dimensions. The input image is stored on the BRAM using the row-major memory layout (ISON1570, 2011), thus the memory address in terms of the image coordinates is calculated using the Equation 5.3. In order to generate one pixel in an image from the pooling layer 1, which has 2x2 kernel, four pixels of an image from the Convolutional layer 1 must be generated. As the convolutional layer 1 sweeps the input image with a 5x5 kernel, 100 pixels (2x2x5x5) from the input image must be processed to generate one pixel in an image from the pooling layer 1. Instead of generating a whole row of an image from the convolutional layer 1 at a time, a 2x2 kernel is generated at a time, which enables that a pixel can be generated in an image from pooling layer 1, as shown in 5.7. This approach considerably improves the system performance.

$$mem\_address = x * width + y \tag{5.3}$$

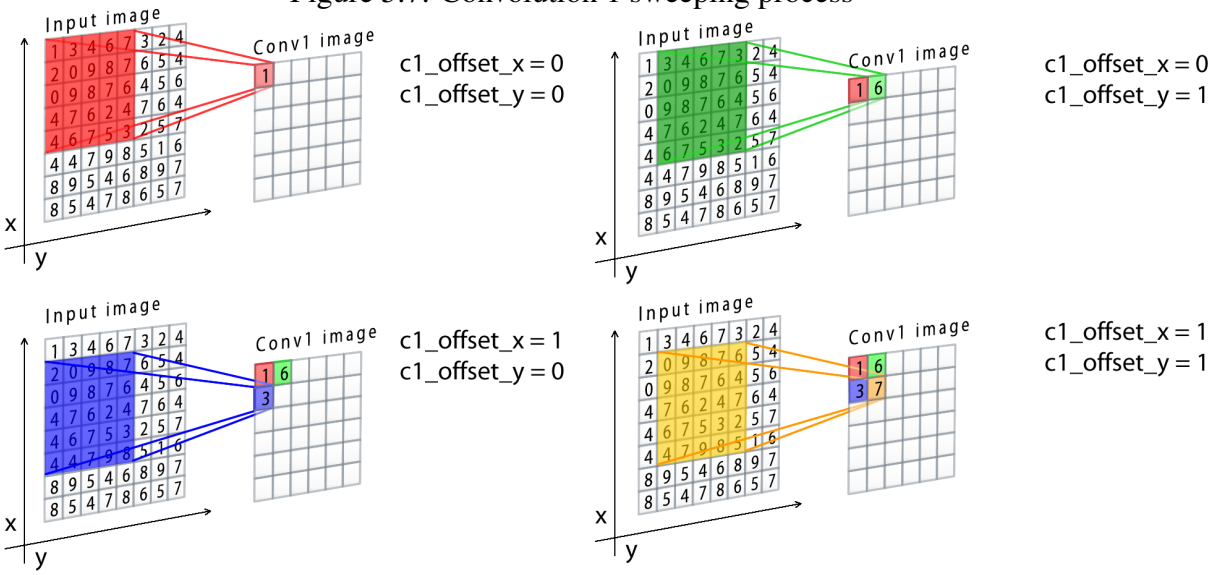
This sweeping process is controlled by the c1_offsets and is the same for all the

Figure 5.6: Convolution-1 control component state machine



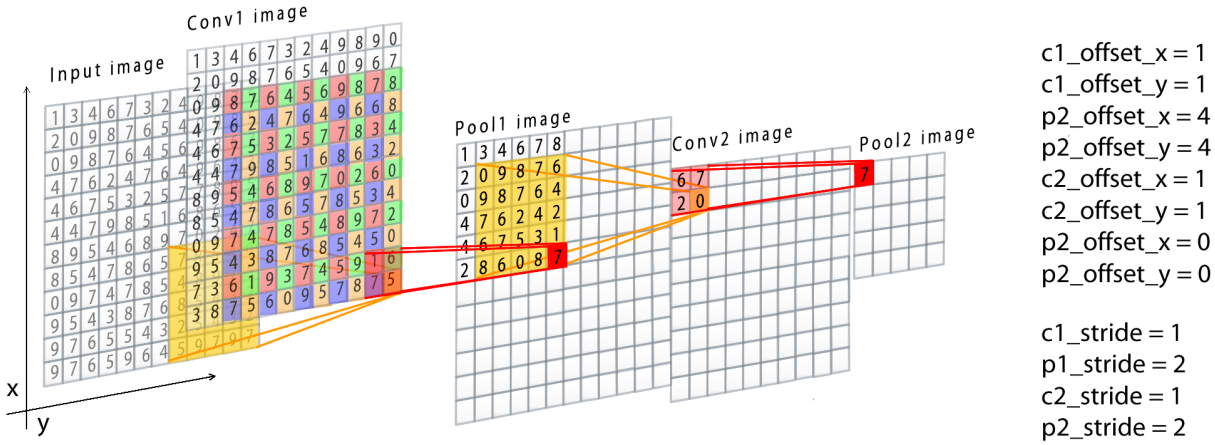
Source: The Author

Figure 5.7: Convolution 1 sweeping process



Source: The Author

Figure 5.8: CNN sweeping process



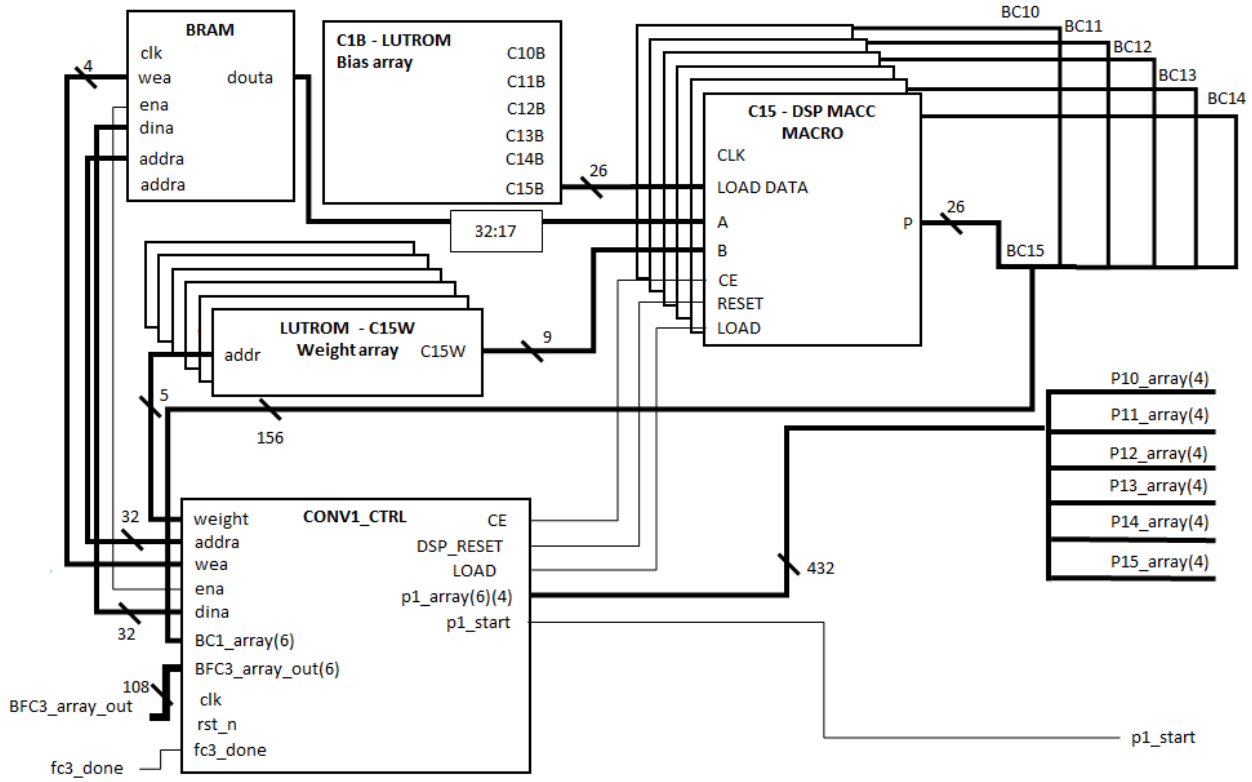
Source: The Author

convolutional and pooling layers, as illustrated in Figure 5.8. The offset of a layer controls the input image addressing in the BRAM. The offset calculation depends on all the previous layer parameters. Consequently, as far a layer is from the input layer (input image stored in BRAM) more complex is its offset calculation, which is controlled by the CONV1_CTRL component. The offsets described in Figure 5.8 were simplified, actually they are products from the convolution and pooling strides starting from the convolution layer 1 to the layer being computed. The counter limits used by the offsets are also products of the convolutional and pooling kernels and strides.

The details of the control and operational part of the convolutional layer 1 is illustrated in Figure 5.9. As it can be seen, one DSP MACC macro for each feature map is instantiated (for simplicity, signals of only one DSP were shown) and each one has its own weight LUTROM. The weight LUTROM output is connected to the "B" input of the DSP that is one of the operands of the MACC MACRO. The weight LUTROM address is controlled by the CONV1_CTRL component that performs the timing-multiplexing in the DSP weight inputs. The bias does not need to be multiplexed, inasmuch as each feature map or Full-connected neuron has only one bias. Therefore, each bias LUTROM output is fixed to a specific DSP "LOAD DATA" input. The CONV1_CTRL component loads the bias into the DSP through the "LOAD" signal and after it finishes a MACC computation, it resets the DSP by the "DSP_RESET" signal". As the Convolutional and Full-connected parameters fractional precision was set to 8 bits, the pixels have 8 bits (1 Byte), only 17 bits out of the 32-bit BRAM output were used as the "A" input of the DSP (1 sign-bit, 8 bits for integer-part and 8 bits for fractional-part).

The BRAM output and weight LUTROM outputs must be synchronously assigned to the DSP inputs by the CONV1_CTRL component so that the correct results

Figure 5.9: Convolution 1 - Control and datapah



Source: The Author

are produced. In order to prepare the receptive field of pooling layer 1 (2x2 kernel), the CONV1_CTRL component receives an array that contains all the DSP outputs in the layer, the "BC1_array" (Blob convolution 1 array). Thus, after 4 convolutional computations, the CONV1_CTRL component reduces the precision of the Convolution 1 output signals, then these values are assigned to the P1_ARRAY (Pooling 1 array), after which it writes a 'high' value to the the p1_start signal. It is also possible to note that the CONV1_CTRL component receives a "FC3_done" (Full-connected-3 done) signal and a "BFC3_ARRAY_OUT" (Blob Full-connected-3 Array Out) signal array. A "high" value in the "FC3_done" signal indicates that the Full-connected-3 layer finished its computation, whereas the "BFC3_ARRAY_OUT" signal array contains the Full-connected-3 layer outputs. When the CONV1_CTRL component reads a "high" value in the FC3_done signal, it changes the "wea" signal to write mode in order to write the values of the "BFC3_ARRAY_OUT" array into the BRAM.

The layer implementations have similar architectures. They have an operational part that can be composed by DSPs, weight LUTROM and bias LUTROM, which are present in Convolutional and Full-connected layers. Alternatively, the operational part can be composed only by Max comparators, which are present in Pooling layers. All the

layers have a control part that remains idle while its "start" input signal is "low". The control part of the layers controls the receptive field of the next layer so that when the computation of the current layer is finished, it writes a "high" value to the "start" signal that is read by the next layer. However, only the CONV1_CTRL component has access to the BRAM. The simplest controllers are the Full-connected control components. As they have one-dimension receptive field, they do not need to control the receptive field of the next layers.

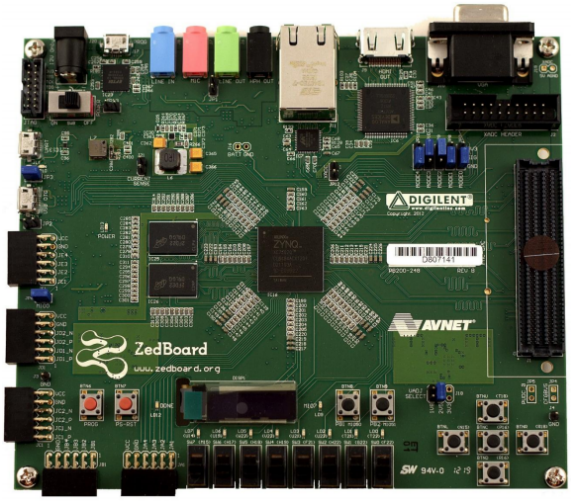
5.3 APSoC Implementation results

The case study CNN was implemented on the Xilinx Zynq-7000 APSoC XC7Z020-CLG484. Figure 5.10 shows the Zedboard that contains the Zynq-7000, whose features are described in the list bellow:

- Xilinx Zynq-7000 AP SoC XC7Z020-CLG484
- Dual-core ARM Cortex™-A9
- 512 MB DDR3
- 256 MB Quad-SPI Flash
- 4 GB SD card
- On-board USB-JTAG Programming
- 10/100/1000 Ethernet
- USB OTG 2.0 and USB-UART
- Analog Devices ADAU1761 SigmaDSP® Stereo, Low Power, 96 kHz, 24-Bit Audio Codec
- Analog Devices ADV7511 High Performance 225 MHz HDMI Transmitter (1080p HDMI, 8-bit VGA, 128x32 OLED)
- PS & PL I/O expansion (FMC, Pmod, XADC)

The CNN resource requirement that does not enable a bigger CNN implementation is the FC weight storage. The Full-connected MLP, as the name implies, has all of its inputs connected, so it considerably increases the weight storage requirement. The number of FC weights is calculated by multiplying the number of neurons from a FC layer times the number of inputs of each neuron from this layer, then summing the weights of all FC layers. The initially proposed CNN model (YANG et al., 2016) has 800 neurons in

Figure 5.10: ZedBoard



Source: (REN et al., 2015)

the first FC layer. Each neuron has a receptive field of 36 feature maps, each of them possessing dimensions of 5x5. Therefore, the first FC layer requires (800x36x5x5) 720000 weights. The second layer has 256 neurons, each of them possessing 800 inputs (the number of neurons from the previous FC layer), totaling (256*800) 204800 weights. The third FC layer has 12 neurons, each of them having 256 inputs (12*256), totaling 3072 weights. Thus, the total number of FC weights is (720000 + 204800 + 3072) 927872.

The number of convolutional weights is calculated by multiplying the number of pixels of the kernel (dimensions) times the number of feature maps from the previous layer times the number of feature maps from the current layer, then summing the weights of all convolutional layers. The convolutional layer 1 has 16 feature maps, each one having 5x5 kernel and a previous layer with 1 feature map (input layer), totaling (5x5x1x16) 400 weights. The convolutional layer 2 has 32 features maps each one possessing 5x5 kernel and with the previous layer having 16 feature maps (5x5x16x32), totaling 12800 weights. The total number of convolutional weights is (400 + 12800) 13200. So, the weight sharing benefit of convolutional layers considerably reduces the weight storage requirement. Adding the number of FC weights to the number of convolutional weights of the CNN model of (YANG et al., 2016), the result is (927872 + 13200) 941072, whereas the total number of weights of the proposed reduced CNN is 18190.

Hence, the reduction of the number of FC neurons is crucial to memory usage reduction. As the FC neurons and convolutional neurons are distributed across the FPGA area, the distributed memory was used (LUT as memory) for their weight storage. Therefore, the initial proposed CNN implementation exceeded the available LUT resources. Although the total number of weights was reduced by 98.03% (from 941072 to 18190),

the LUT utilization comprises 59.14% of the available APSoC LUTs, as will be shown in the following results.

DSP slices were used because they are efficient for digital signal processing implementations, like deep learning, in virtue of their dedicated routing signals and arithmetic blocks, reducing the use of LUTs and Carry Logic of the Configurable Logic Blocks (CLBs), reserving these resources for other generic usage (FU et al., 2017). As shown in Table 5.2, even with the DSP usage the LUT utilization comprises 59.14% of the available LUTs. A great part out of these resources was used to the weight storage, as previously explained. Given the fact that more complex CNNs are usually implemented on high-end APSoC platforms, like the Xilinx Ultrascale family (FU et al., 2017), the case study CNN is suitable for the available mid-range APSoC platform.

Table 5.2: FPGA resource utilization

Resource	Utilization	Available	Utilization %
LUT	31463	53200	59.14%
LUTRAM	2132	17400	12.25%
FF	27234	106400	25.60%
BRAM	4.50	140	3.21%
DSP	94	220	42.73%
IO	12	200	6.00%
BUFG	2	32	6.25%

Source: The author

The complexity involved in sweeping the input image stored in the BRAM taking into account the successive layers gave good performance results, which enabled a 313 FPS rate on 32x32 images, as shown in Table 5.3. In addition, the DSP usage also avoids that timing constraints are not met due to their dedicated routing signals (Depending on the routing through and between the CLBs, the timing constraints may not be met due to the delay associated to long paths).

The Softmax and Max execution time in the ARM processor are in the range of microseconds, whereas the CNN execution time is the range of milliseconds. Even with the sums of exponential being computed by serial instructions instead of dedicated hardware, the Softmax computation achieved excellent timing results, having little contribution in the total execution time. Thus, other computer vision tasks can be explored in the ARM processor in order to take advantage of the benefits of each part of the APSoC platform (PS and PL).

In order to predict the performance of greater CNNs, the performance scalability of the APSoC implementation is given in terms of CNN model parameters. Firstly,

Table 5.3: Performance results

Description	Value
PL Frequency	100 MHz
PL Clock Cycles	303,170
PL Execution time	3.03170 ms
PL Execution time	3.03170 ms
PS Frequency	666.66 MHz
PS Softmax Clock cycles	135480
PS Softmax Execution time	203.42 us
PS Max Clock cycles	22116
PS Max Execution times	33.21 us
PS Total Clock cycles	157596
PS Total execution time	236,63 us
Total execution time	3.26833 ms

Source: The author

visualize some important acronyms defined in Table 5.4.

Table 5.4: CNN feature abbreviations

Abbreviation	description
cv1k	Convolution 1 kernel size (squared)
c1fm	Convolution 1 feature maps
p1k	Pooling 1 kernel size (squared)
c2k	Convolution 2 kernel size (squared)
c2fm	Convolution 2 feature maps
p2k	Pooling 2 kernel size (squared)
p2i	Pooling 2 image dimension (squared)
fc1n	Number of neurons of Full-connected layer 1
fc2n	Number of neurons of Full-connected layer 2
fc3n	Number of neurons of Full-connected layer 3
ctrl_overhead	Performance overhead due to the control

Source: The author

The performance of a CNN implemented by the proposed tool is defined by Equation 5.6 that is the sum of the clock cycles taken by the effective computation (Equation 5.4) and the performance overheard associated to the timing-multiplexing control (Equation 5.5).

$$clock\_cycles = (cv1k \times p1k \times c2k \times p2k \times p2i) + (c2k \times c1fm) + (p2i \times c2fm) + fc1n + fc2n \tag{5.4}$$

$$ctrl\_overhead = ((((5)p1k + 1)c2k + 1)p2k + 1) \times p2i + 16 \tag{5.5}$$

$$total\_clock\_cycles = clock\_cycles + ctrl\_overhead \tag{5.6}$$

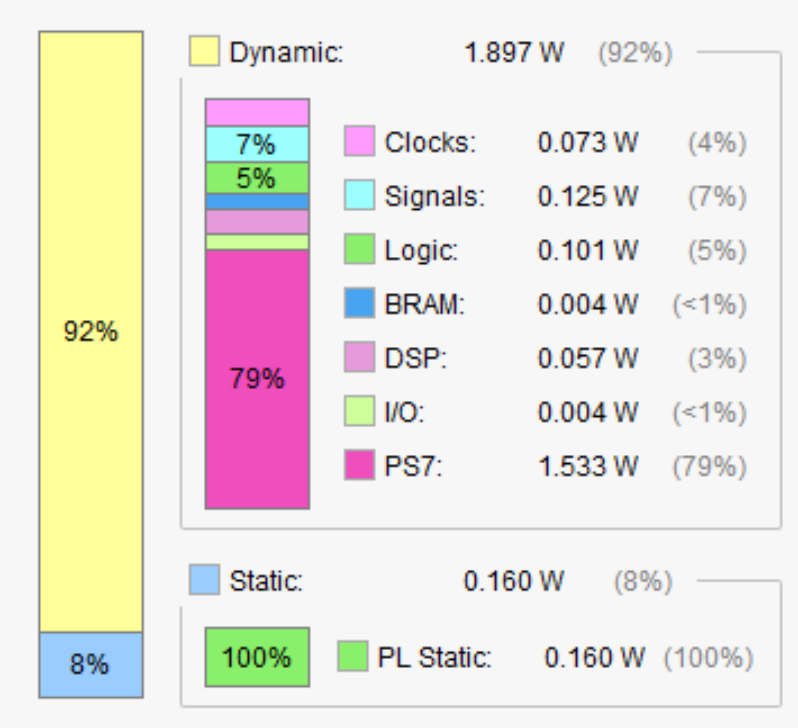
Using Equation 5.6 to predict the performance of the initially proposed CNN (YANG et al., 2016) that classifies 12 sub-classes, it is possible to know that it would take only 304,897 clock cycles. Therefore, if an APSoC platform which satisfies the weight storage (more LUTs are required) and the DSP resource requirement is used, the performance change will be negligible in contrast with the reduced CNN that classifies only 6 sub-classes (303,170 clock cycles). Consequently, the CNN execution time running at 100 MHz in the PL (3.04 ms) will be lower than or almost equal to the related work which performs the classification in 173ms (KASSANI; TEOH, 2017), 40 ms (WANG et al., 2013), 33ms (LIU et al., 2016), 11.4 ms (CIREŞAN et al., 2012), and 3ms (YANG et al., 2016). Comparison with the work that implemented ADAS on an ASIC (LEE et al., 2017) and on an APSoC (HAN; ORUKLU, 2014) platform were not done because this work does not separate the execution time of the classification and detection phase.

However, the FPGA resource utilization of the proposed timing-multiplexing architecture was restricted in order to reserve some resources for the Fault injection setup, which will be described in the next chapter. So, if almost 100% of the FPGA resources are used, the CNN architecture will be more parallelized and the performance will be even better.

As shown in Figure 5.11, summing the dynamic (1.897 W) and static (0.160 W) power consumption, the total power consumption of the proposed CNN implementation on the target APSOC is about 2.05 W. The Drive PX-2 embedded GPU in its minimum operation, with one passively cooled mobile processor, operates at 10 W. If the multi-chip configuration is enabled, with four high-performance AI processors, the power consumption reaches 250 W (HUANG, 2016)(NVIDIA, 2016). Therefore, the proposed CNN implemented on the target APSoC consumes less power than an embedded GPU in its minimum operation.

As shown in dark pink in Figure 5.11, most of the power consumption is due to the PS. This is expected, because the ARM processor operates at a 666.6 MHz frequency while FPGA operates at a 100 MHz frequency. Although the PS part of the APSoC was only used to perform the softmax function, it is a good practice to use the PS to make decisions and assessment for ADAS tasks. Beyond the ARM processor, the PL contains Advanced Microcontroller Bus Architecture (AMBA) interconnections and several buses such as Serial Peripheral Interface (SPI), Controller Area Network (CAN) and

Figure 5.11: Total SoC Power consumption

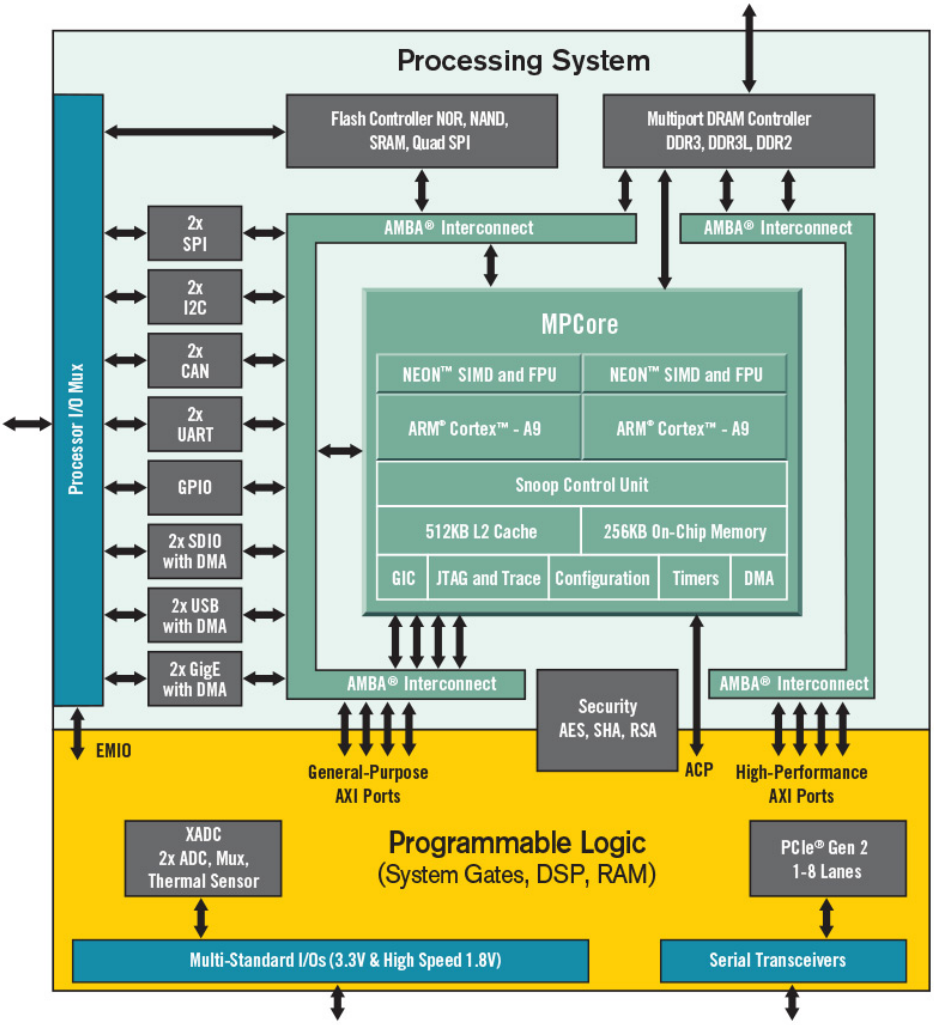


Source: The author (generated by Xilinx Vivado 2017.1)

Inter-Integrated Circuit (I2C), as shown in Figure 5.12 (XILINX-ZYNQ, 2017). ADAS or autonomous car systems can receive information by a vision system implemented on the PL by using the "AMBA interconnect" and then actuate the car using the CAN protocol. This system level control can be developed in software using the software programmability of the ARM-based processor. So the PS is the main part of a real ADAS or autonomous car system (XILINX-AUTOMOTIVE, 2016).

The advantage of the PL over the PS is its parallelism. As deep learning algorithms are implemented using mathematical operations that can be computed in parallel, it is a common practice to implement objects recognition (as the traffic-sign recognition) on the PL part for pixel-level analysis (XILINX-AUTOMOTIVE, 2016).

Figure 5.12: Zynq-7000 block diagram



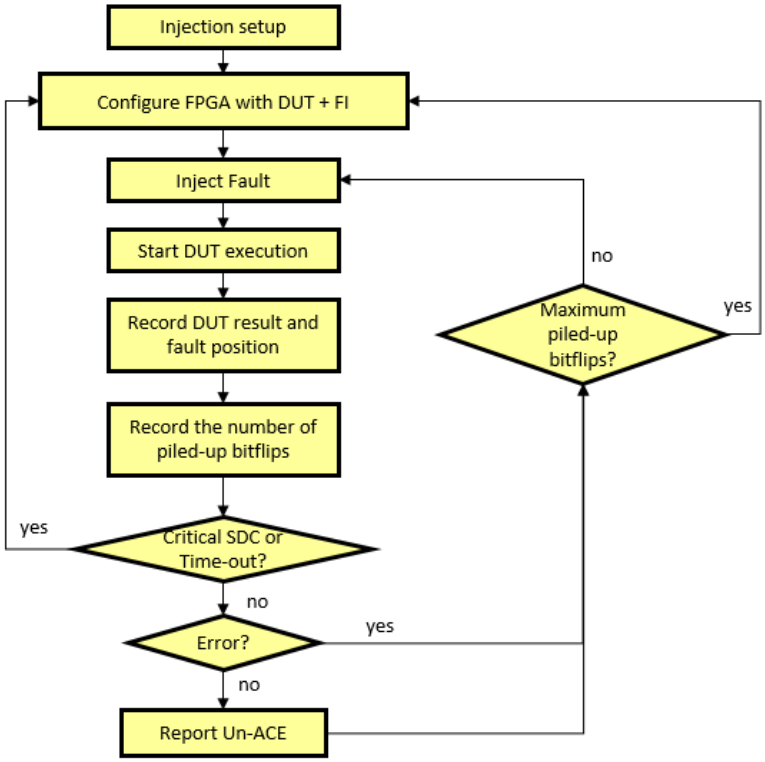
Source: (XILINX-ZYNQ, 2017)

6 FAULT INJECTION BY EMULATION

As the CNN was implemented on a FPGA of an APSoC, its peculiar radiation effects must be taken into account. The persistent effects of a fault in the configuration bits is the main concern in a FPGA reliability analysis, given that the upsets in Flip-Flops are usually transient effects (KASTENSMIDT; CARRO; REIS, 2006). There are three ways of analyzing the reliability of a FPGA design or another integrated circuit, it can be performed by simulation, emulation or radiation experiments. Radiation experiments can be performed at ground level, high altitudes and space level. Radiation experiments at ground level can be carried out by particle-beam (ZIEGLER et al., 1996) or laser testing (POUGET et al., 2001) and present an acceleration of the real-world scenario, while radiation experiments at high altitudes (O'GORMAN et al., 1996) or at space level (VELAZCO et al., 1999) are the real-world scenario, however these experiments can take from months to years. Therefore, radiation experiments are the most accurate tests. A reliability analysis by emulation can artificially cause the same fault in the real circuit that can take place by a radiation experiment and can be done using one (NAZAR; CARRO, 2012)(BERNARDI et al., 2004), two (ALDERIGHI et al., 2007) or three (WIRTHLIN et al., 2003) FPGA boards. A simulation, in turn, tries to imitate both the real-world circuit and radiation effect using either Technology Computer Aided Design (TCAD) (GADLAGE et al., 2010) or processor simulator (LI et al., 2008). In addition to the ways of analyzing the reliability, there are also many architecture levels such as device, component, block and system. Both reliability analysis types and architecture levels have pros and cons regarding their precision, cost, controllability, observability, performance and other issues (NAZAR; CARRO, 2012).

Radiation experiments are the most accurate tests but they are very expensive. They may have very low controllability and observability depending on the test setup. The time required for radiation experiments can be months, on space and high altitude expositions, or several minutes or hours using particle accelerators. Laser testing can provide a better controlability during the fault injection. The simulation has the highest controllability and observability and, in general, its performance is inversely proportional to its accuracy. The simulation of faults can be performed by computing simulation or emulation in hardware platforms. The cost depends on which platform it is performed which in turn is also related to its performance (PCs, data-centers, clusters of GPUs, FPGAs, supercomputers and so on). The emulation and computing simulation of faults normally

Figure 6.1: Piled-up fault injection flowchart



Source: The author

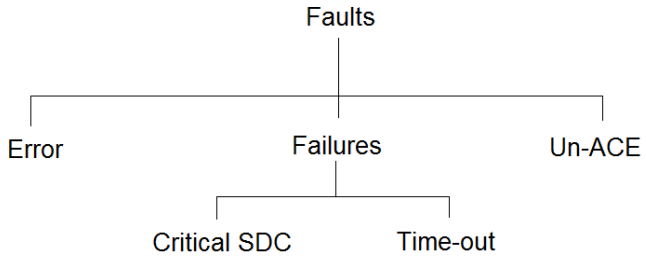
present a very high controlability and observability. It can have high accuracy depending on how the fault is emulated, and in general for the same platform used in a simulation, its performance is considerably lower than in a simulation, thus enabling implementations in less expensive platforms than in simulation (NAZAR; CARRO, 2012).

The architecture level pros and cons depend on the goal of the test, because they depend on the reliability test, platforms and other issues. However, in general, the higher the abstraction level of the test, the easier it is to extrapolate the result to a system effect. A reliability test and architecture level is strongly dependent on its goal (military, academic, commercial and so on), thus for academic purposes Fault Injection (FI) by emulation is a suitable approach (HSUEH; TSAI; IYER, 1997)(ZIADE et al., 2004).

There are many ways of injecting faults by emulation in FPGA configuration bits and it depends on what is the fault injection purpose. If someone wants to know how many critical bits out of the essential bits a design have, an exhaustive fault injection that encompasses all the configuration bits used by a design is indicated. Alternatively, if someone wants to estimate the statistical reliability of a FPGA design, that is, how reliable a FPGA design under random distributed faults is, a piled-up random fault injection is suitable. The random piled-up fault injection flowchart is illustrated in Figure 6.1.

First, the fault injection setup is initialized, then the FPGA is configured with the

Figure 6.2: Faulty hierarchy



Source: The Author

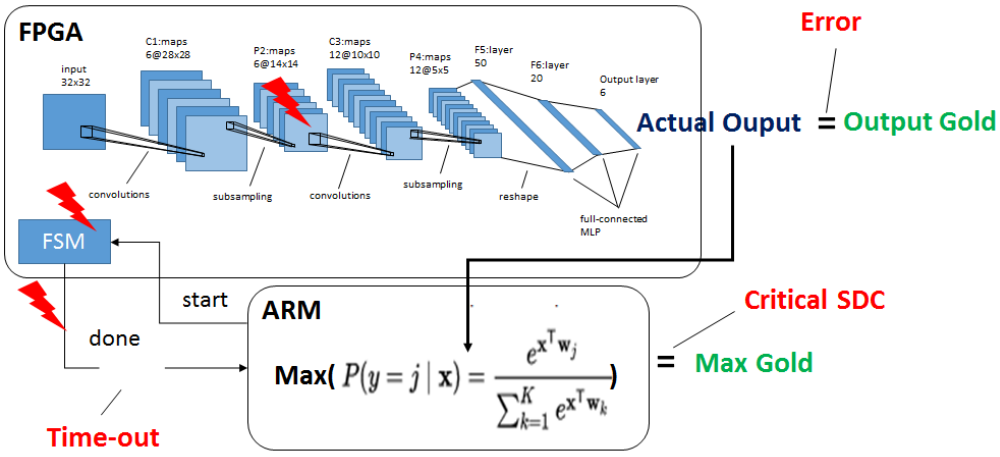
Design Under Test (DUT) and the Fault injection (FI) setup. After which, a fault is injected in a random position of the design configuration bits. Then, the Design Under Test (DUT) is executed, so the fault position and the DUT result, as well as the current number of piled-up faults, are registered in a log file. If a failure (either critical Silent Data Corruption (SDC) or time-out) does not occur, the fault injection process continues until the maximum number of injected faults is reached. If a failure occurs or the maximum number of injected fault is reached, the FPGA is re-configured and the process is restarted. However, errors and un-Architecturally Correct Execution (un-ACE) do not stop the piled-up fault injection. The critical SDC, time-out and errors, as well as the FI setup, will be explained in the next sections.

6.1 Failure Model

Before proceeding with the explanation of the fault classification, some soft-error terminologies must be defined. Some bits out of a hardware design (PL) or a software (PS) are responsible for the Architecturally Correct Execution (ACE) of the respective hardware design or software. On the other hand, the bits that do not affect the system output are called un-ACE bits (MUKHERJEE et al., 2003). Errors are often further classified as detected and undetected. Detected errors are referred to as Detected Unrecoverable Errors (DUEs), whereas the undetected errors are referred to as Silent Data Corruptions (SDCs) (MUKHERJEE et al., 2003). Not always a SDC causes a function "failure" but when it occurs the SDC is called as "critical SDC". When a SDC does not cause a function "failure" it is called as "error". If the system does not reply after its specified execution time, a "time-out" occurs and since the system does not perform its function it is classified as a "failure". The faulty hierarchy is illustrated in Figure 6.2.

Considering the CNN topology implemented on the FPGA, there are three effects

Figure 6.3: Failure model APSoC CNN



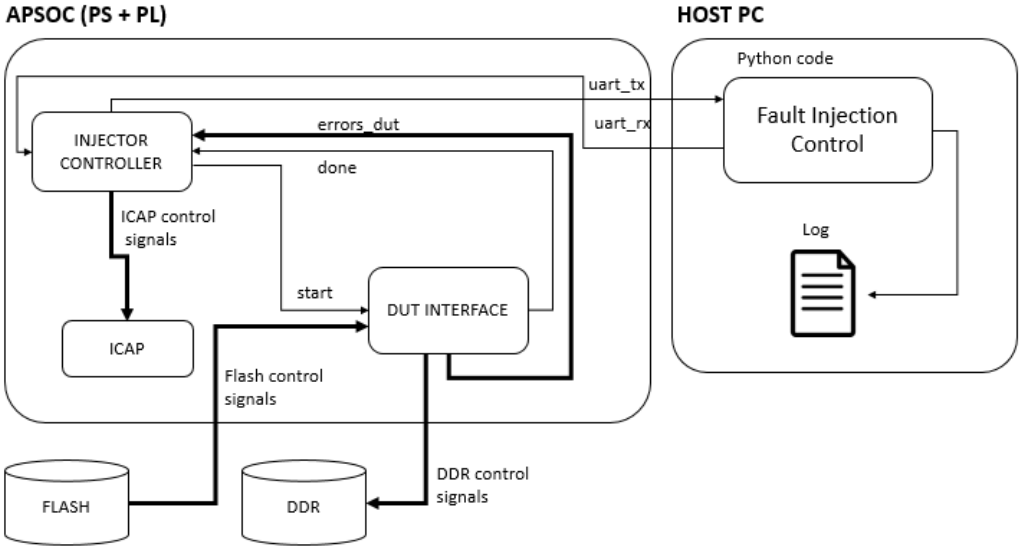
Source: The Author

(Error, failure, un-ACE) that can take place due to a fault in the configuration bits. One fault across the CNN can be propagated to the output so that the result is different from that expected, but after the softmax computation and max computation the maximum value is still preserved. As explained before, this type of effect is classified as an "error". Although the system output is corrupted (SDC), the system functionality is kept. Alternatively, the system output can be corrupted so that after the softmax and max computation, the maximum value differs from that expected. This "critical SDC" is classified as a "failure" because the system does not perform its main functionality. Due the fact that the CNN was implemented in a multiplexed way instead of a full-parallel way, it requires that after a certain number of clock cycles the CNN informs the ARM processor that the computation is done and sends the results. Thus, if a fault in the configuration bits causes an "open" in a wire that sends the done signal, or if an offset counter that defines whether a state transition will occur or not, as well as if other faults occur, the ARM processor never will receive a done signal. When a system does not perform its specified function after its specified execution time, a "time-out" occurs, and this effect is classified as "failure". When none of these effects occur, the injected fault causes an un-ACE effect. An illustration showing how a SEU (Thunder) can cause different type of effects is given in Figure 6.3.

6.2 Experimental Setup

The experimental setup is composed by a Zedboard APSoC and a host PC, as illustrated in Figure 6.4. The Host PC executes an application in python that randomizes

Figure 6.4: Fault injection setup



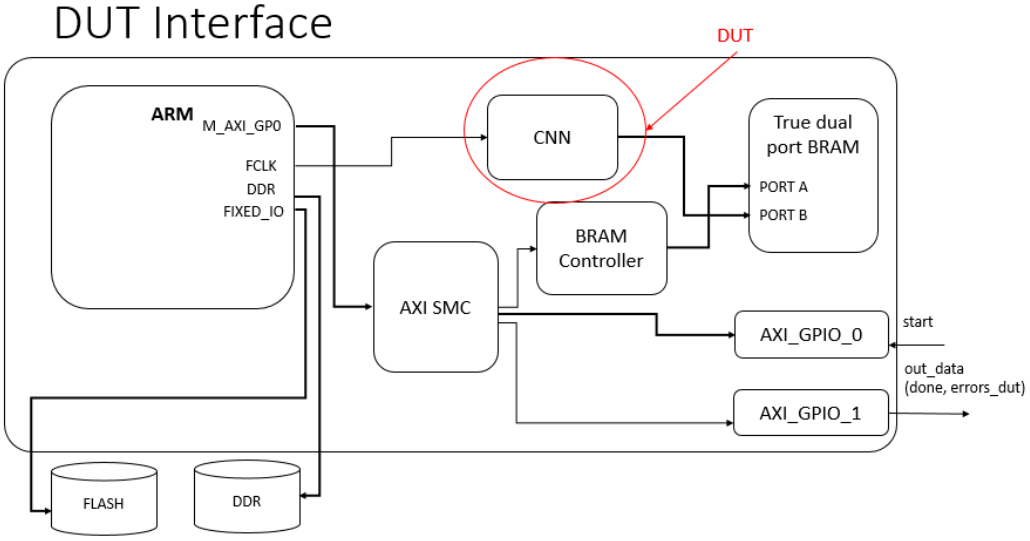
Source: The author

the configuration bit position of the design, and then sends a configuration frame to the FPGA by a Universal Asynchronous Receiver/Transmitter (UART) port emulated with a Universal Serial Bus (USB) connection. After the fault is injected, the DUT is executed, and the result is received, the application writes the FI results in a log file, and then resets and re-configures the FPGA when it is necessary. The APSoC contains the ARM processor (PS) and the FPGA (PL) which, in turn, contains the injector controller component, the Internal Configuration Access Port (ICAP) and the DUT interface component. There are also additional DDR and QUAD SPI FLASH external memories.

The injector controller receives the commands from the Python application, running at the host PC by an UART connection, then sends the frame and instruction to the ICAP in order to inject the fault. The ICAP is the embedded component in the Xilinx FPGAs responsible for partial-reconfiguration of the configuration bits. After the ICAP to perform the fault injection, the injector controller starts the DUT sending a "start" signal to the DUT interface, then waits for a "done" signal from the same as well as the error classification. The DUT interface is illustrated in Figure 6.5. The "start" and "out_data" signals, coming from and going to the injector controller, communicate with the ARM processor through AXI General Purpose Input/Output (GPIO) connections using the AXI Static Memory Controller (SMC) bus. The ARM processor communicates with the external FLASH memory in order to re-configure the FPGA when necessary.

These were the only modifications in the design described in section 5.2, the remaining configurations are the same described in Figure 5.3. Actually, the APSoC design was implemented aiming at a fault injection experimental setup. The ARM application

Figure 6.5: DUT interface component expansion

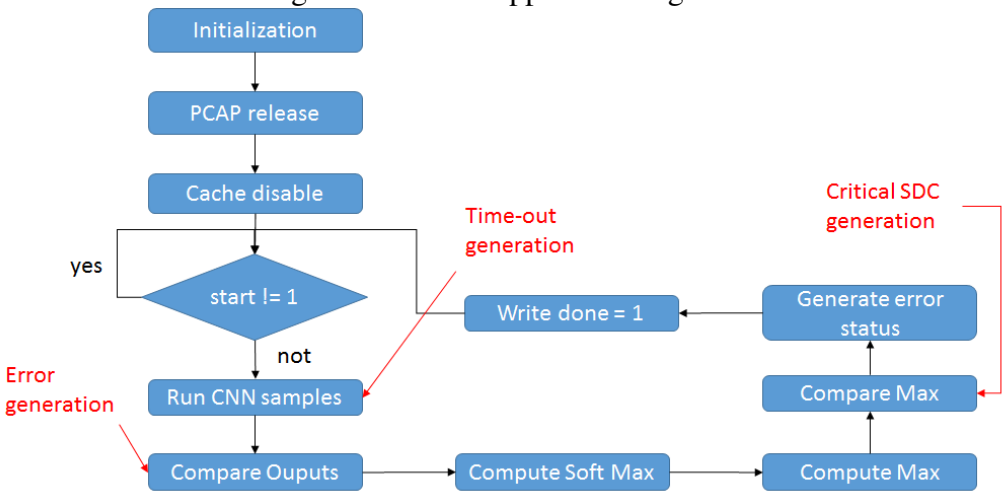


Source: The author

simplified flowchart is illustrated in Figure 6.6. First, the application is initialized, then the Processor Configuration Access Port (PCAP) is disabled in order to enable the ICAP usage. After which, all the cache levels are disabled to avoid data corruption. Then, the code stays in an empty loop while the start signal is 0, otherwise the CNN is executed classifying 6 images (one per class). If the CNN does not write a "done" into the BRAM after its execution time, a "time-out" is signalized. If a time-out does not happen, the FC outputs are compared with the Gold ones. If the values are not identical, an "error" is generated. After which, the FC signals are converted to float, and the Softmax function is computed. If the index of the maximum value of the Softmax is not identical to the Gold one, a critical SDC is generated. Then, a "done" signal, as well as the error classification, are sent by an output GPIO port and the loop restarts. This application was executed in Bare Metal at the ARM processor.

Lastly, the defined APSoC floorplanning is shown in Figure 6.7. Xilinx APSoC floorplanning is divided into top and bottom regions that can comprise different numbers of rows. The address of a row on the bottom region increments from center to bottom and of a row on the top region increments from center to top. Different than expected, Xilinx referred to a block of many rows as "row". Thus, the Zedboard Zynq-7000 floorplanning is divided into 3 rows, one row (row 0) lies on the top region and the other two rows (row 0 and row 1) are situated on the bottom region. The vertical lines are the columns and the black block on the left top corner is the ARM processor. The minimum addressable portion of configuration bits is the "frame". The frames comprise 101 words of 32 bits by which are addressable by their bit position. Through the frame address and the bit position

Figure 6.6: ARM application algorithm

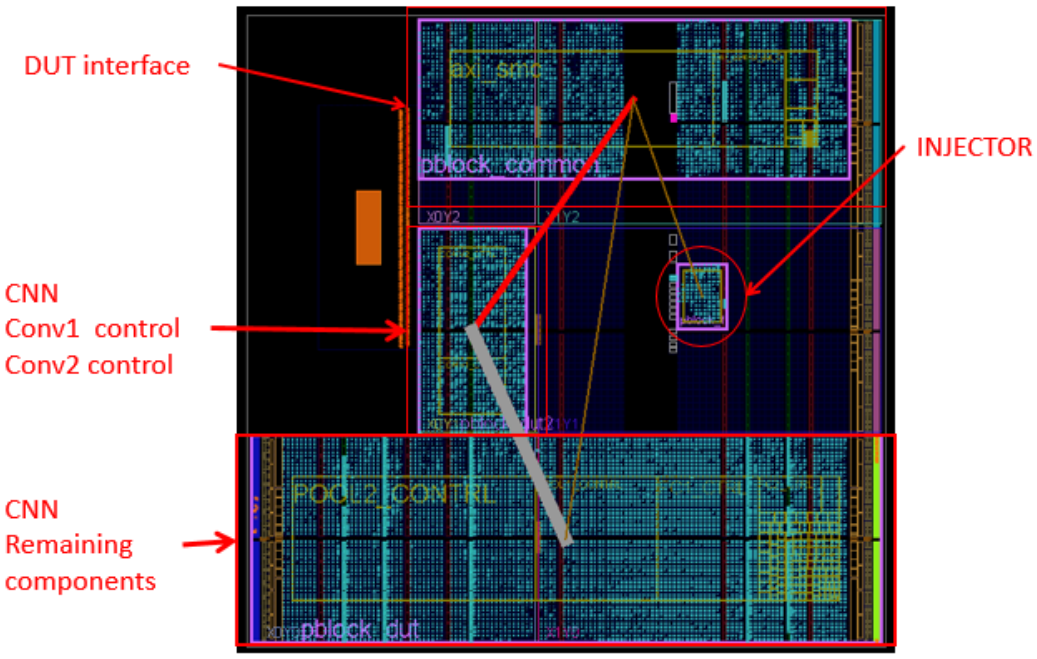


Source: The author

it is possible to choose at which specific position of the implemented design the fault will be injected. Thus, the frames are addressable by their bottom/top regions, their row, their major (column) and minor (sub-columns). Unfortunately, the CLB/DSP horizontal lines are not directly associated to the minors and Xilinx does not provide the information with regards to how the CLB/DSP lines are coded as minors. Therefore, the DUT components must be placed over an entire row on the bottom or top region and the columns must be limited by a column range.

Due to the high area utilization of the CNN design, it had to be divided into two physical blocks (pblocks), one on the left of the bottom row 0 and another in the entire bottom row 1. The injector controller is isolated from the DUT regions and strategically placed near the ICAP, the remaining components of the design that do not belong to the DUT are isolated from the DUT regions in the top row 0 with a reliable level of distance from the DUT region on the bottom row 0. This isolation is necessary in order to avoid fault injection in injector routing signals that could cross the DUT region if the injector was placed near the DUT region. The DUT region placed on the bottom row 0 contains the CONV1_CTRL and the CONV2_CTRL components, whereas the DUT region placed on the bottom row 1 contains the remaining components of the CNN design.

Figure 6.7: Implemented design floorplanning



Source: The author

7 RELIABILITY RESULTS

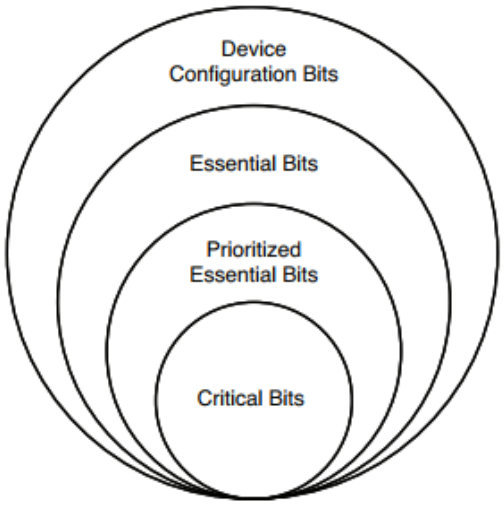
As explained in the previous chapter, the fault effects were classified into un-ACE, errors and failures. The critical SDCs and time-outs are found in the set of failures. When critical SDCs occur, the correct classification is changed and when time-outs occur the CNN does not perform the computation in its specific execution time. Firstly, the higher level hierarchy effects (un-ACE, errors and failures) will be discussed, later only the failures will be analyzed, then the number of incorrect classifications and lastly the reliability curve of the CNN.

Before performing the analysis in the effect of piled-up faults in the configurations bits, some definitions concerning the PL configuration bits must be introduced. The Zynq-7000 APSoC XC7Z020 has 25.6E6 configuration bits. However, a subset of these configuration bits is used by the user design. This subset is called as "essential bits". Xilinx has an algorithm to identify the essential bits out of the configuration bits. This information is obtained by setting constraints in the XDC file, and can be found in a log file after the bitstream is generated. Essential bits enable accurate evaluations of the reliability of a design implemented on FPGA (LE, 2012). Xilinx also has a tool to filter the essential bits more important in the design, these bits are called as "prioritized essential bits". A specific block of the design can be defined as prioritized and only its used configuration bits will be defined as essential bits enabling hardening process in specific blocks, and then reducing the area overhead associated with spatial redundancy techniques. The bits out of the essentials that once flipped cause a function "failure" are called as "critical bits". Therefore, the more critical bits the design possesses, the more sensitive is the design. (LE, 2012). The device configuration, essential, prioritized essential and critical bit sets are illustrated in Figure 7.1.

The proposed CNN uses 5.73E6 essential bits out of the configuration bits of the device (22.31%). However, the CNN was isolated in physical blocks that contain 8.18E6 configuration bits. Thus, only the configuration bits used by the DUT physical blocks must be taken into account in order to perform an accurate reliability analysis. Therefore, the CNN essential bits comprise about 57.33% out of the physical block configurations bits. The critical bits percentage, which will be discussed later, must be calculated over the essential bits.

As the function of the CNN design is to classify traffic-signs in a specific execution time, one configuration bit is called critical when the index of the maximum value of the

Figure 7.1: Relationship of Device Configuration Bits, Essential Bits, Prioritized Essential Bits, and Critical Bits



Source: (LE, 2012)

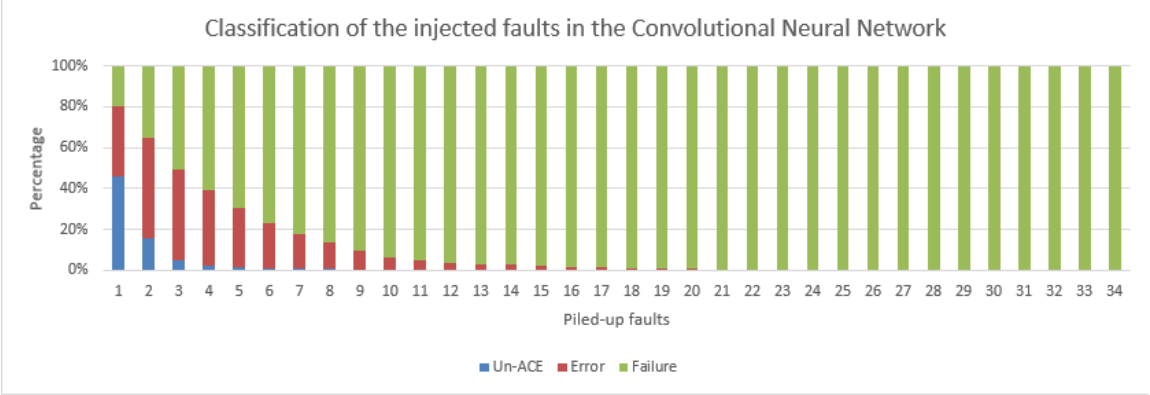
softmax vector is different from the gold one or when the maximum value is not computed in its required execution time. Even if an upset in a given configuration bit changes the output of the Full-connected neurons, as long as it does not change the correct traffic-sign classification (maximum value of the softmax vector), it will not be classified as a critical bit.

The critical bits of a design can be found by exhaustive fault injection by emulation. Unfortunately, each fault injection and result verification take about 200 ms due to the CNN execution time for all the samples, the communication and verification execution time. Thus, injecting faults in 8.18E6 configuration bits will take about 18.95 days. Moreover, when a bit-flip occurs in a bit that transforms a LUT configured as memory into a shift-register, the process of hard resetting the device, reconfiguring the FPGA using the QUADS-PI flash and loading the application code on ARM processor takes about 4 seconds, increasing the total fault injection time. Therefore, the reliability analysis was performed by random fault injection which provides approximate results, on the other hand it is considerably less time-consuming than exhaustive fault injection (NAZAR; CARRO, 2012).

The percentage of un-ACE, errors and failures versus the piled-up faults is plotted in Figure 7.2. 1000 piled-up fault injection campaigns were performed, where the maximum number of piled-up faults was set to 300. Nevertheless, it will be shown in the results that the piled-up faults do not reach this value because when a failure occurs the campaign is ended.

As it can be seen in Figure 7.2 when one fault is injected (SEU), the un-ACE

Figure 7.2: Classification of the injected faults in the Convolutional Neural Network



Source: The Author

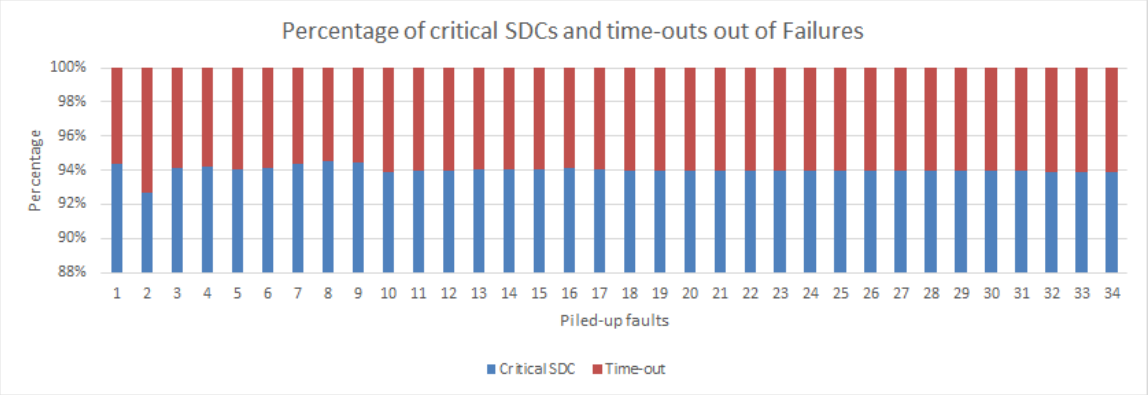
effects comprise 47.3% out of the fault injections campaigns and this percentage is exponentially decreased with a high slope as the faults are piled-up. Un-ACE effects occur up to 31 faults are piled-up. Indeed, un-ACE under many piled-up faults is pretty unlikely (0.1%), provided that from 6 piled-up faults onwards the un-ACE probability of happening is less than 1%.

Regarding the errors under one injected fault (SEU), their percentage comprises 33.2% out of the fault injection campaigns and it dominates the portion of ACE effects (errors + failures). However, as the faults are piled-up, the portion of errors is decreased, whereas the portion of failures is increased. This difference is exponentially increased so that after 20 piled-up faults, approximately all the ACE effects are failures. The errors occur up to 34 faults are piled-up but it has an extremely little probability of happening (0.1%).

These results imply that the proposed CNN has a high number of critical bits among the essential bits, consequently its susceptibility is rapidly increased with the piled-up faults in the essential bits. Once the essential bits comprise about 57.33% out of the physical block configuration bits and 52.7% (100 - 47.3) of the configuration bits injected are ACE bits, approximately 91.92% (52.7/57.33) out of the essential bits are able to cause a difference in the system output. Fortunately, instead of the portion of failures, the portion of errors will not change the correct traffic-sign classification, because they do not change the maximum value of the softmax vector.

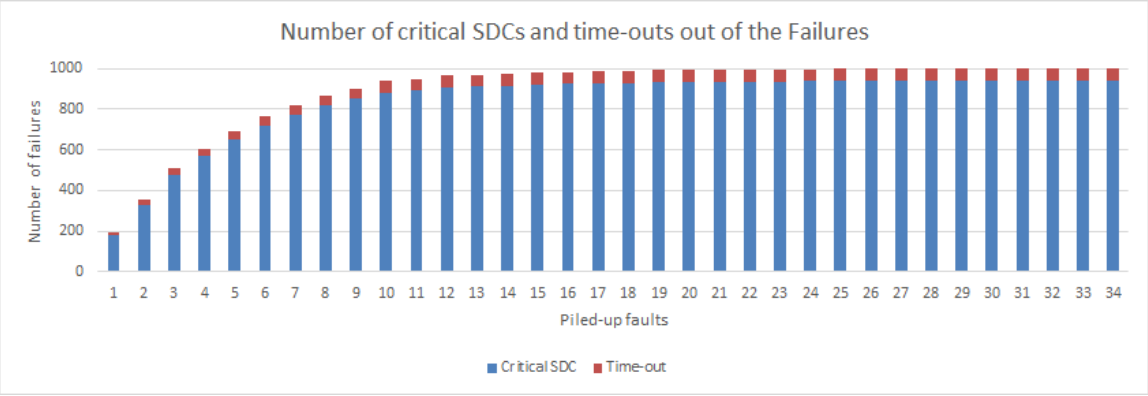
Concerning failures, the percentage of critical SDCs and time-outs out of the failures versus the piled-up faults are shown in Figure 7.3. It is possible to note that regardless of the number of piled-up faults, the critical SDCs and time-outs percentage practically remain the same, the critical SDC percentage varies around 92% to 94% out of the failures. It happens due to an architecture feature of the design. The time-outs are caused

Figure 7.3: Percentage of critical SDCs and time-outs out of Failures



Source: The Author

Figure 7.4: Number of critical SDCs and time-outs out of the Failures



Source: The Author

by upsets in LUTs that implement the logic of the state machines, or routing signals used by these state-machines, consequently the CNN does not complete the computation and never sends the "done" signal. Given the fact that these resources comprise a little portion of the total design resources, it is pretty unlikely that the faults are injected in the configuration bits associated to these resources. Rather, CNNs use a lot of LUTs to store the Full-connected weights and many routing signals to connect layer outputs to layer inputs. An upset in these resources does not affect the state of the system, but only corrupts the system output. Hence, the critical SDCs are more likely to occur.

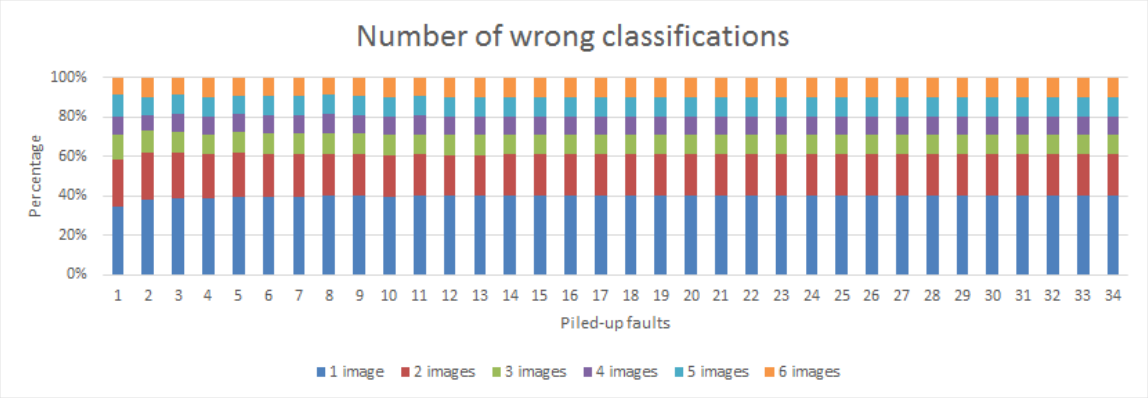
As shown in Table 7.1 and Figure 7.4, although the percentage of critical SDCs and time-outs out of the failures almost remains the same, the number of SDCs and time-outs is increased with the number of piled-up faults.

It is also important to know how many images from the dataset were incorrectly classified in order to analyze the impact of the failures. The dataset contains one image per class, so from one to six images can be wrongly classified. When a critical SDC occurs, there can be different numbers of wrong classifications whereas when a time-out occurs, all the images from the dataset (6 images) are wrongly classified. The percentage

Table 7.1: Number of critical SDCs and time-outs out of the Failures

Piled-up faults	**Critical SDC**	**Time-out**	**Failure**
1	184	11	195
2	329	26	355
3	478	30	508
4	571	35	606
5	651	41	692
6	721	45	766
7	774	46	820
8	816	47	863
9	852	50	902
10	880	57	937
11	892	57	949
12	908	58	966
13	912	58	970
14	915	58	973
15	921	58	979
16	925	58	983
17	927	59	986
18	930	60	990
19	934	60	994
20	934	60	994
21	935	60	995
22	936	60	996
23	936	60	996
24	937	60	997
25	938	60	998
26	938	60	998
27	938	60	998
28	938	60	998
29	938	60	998
30	938	60	998
31	938	60	998
32	938	61	999
33	938	61	999
34	939	61	1000

Figure 7.5: Number of wrong classifications



Source: The Author

of failures by which from one to six images are wrongly classified versus the piled-up faults is plotted in Figure 7.5.

Regardless the number of piled-up faults, the percentage of each number of wrong classifications almost remains the same. Only one image was wrongly classified in about 40% out of the failures, 2 images were wrongly classified in about 20% and 3, 4, 5 and 6 images were wrongly classified in approximately 10%. As upsets in the PL configuration bits are persistent effects, multiple wrong classifications are expected instead of a single wrong classification. However, in 40% of the cases, only 1 image is wrongly classified and the other five are correctly classified. This may occurs, because one image from the dataset may be the most difficult image of being classified. Thus, it is pretty likely that this image will be wrongly classified first. As discussed in sub-section 4.2.2, there is a great probability of this image being from the "sub-class 0". The probability of 2 images being wrongly classified is about 20%, so an image can be the second most difficult image of being classified. As the "sub-class 5" has the second worst accuracy, the image from this sub-class is the most probable of being the second most difficult image of being classified. Lastly, the probabilities of 3 to 6 images being wrongly classified are equally about 10%, and to this extent, four images may be equally difficult of being classified. The absolute number of occurrences for each case is shown in Table 7.2 and Figure 7.6. It is evident that although the percentage of each case nearly remains the same, the number of failure occurrences are increased with the number of piled-up faults.

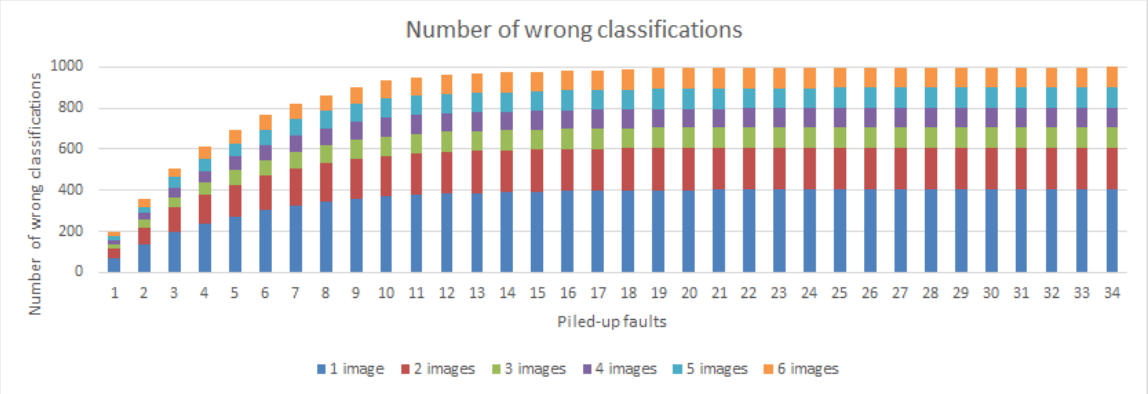
The probability that a visible system error will occur given a bit flip in a storage cell is expressed by the Architectural Vulnerability Factor (AVF) (MUKHERJEE et al., 2003). Thus, the reliability can be expressed by the complement of the AVF (100% - AVF). The reliability curve when considering failures, is plotted in Figure 7.7.

As it can be seen in Figure 7.7, the reliability of the CNN considering failures

Table 7.2: Number of wrong classifications (absolute values)

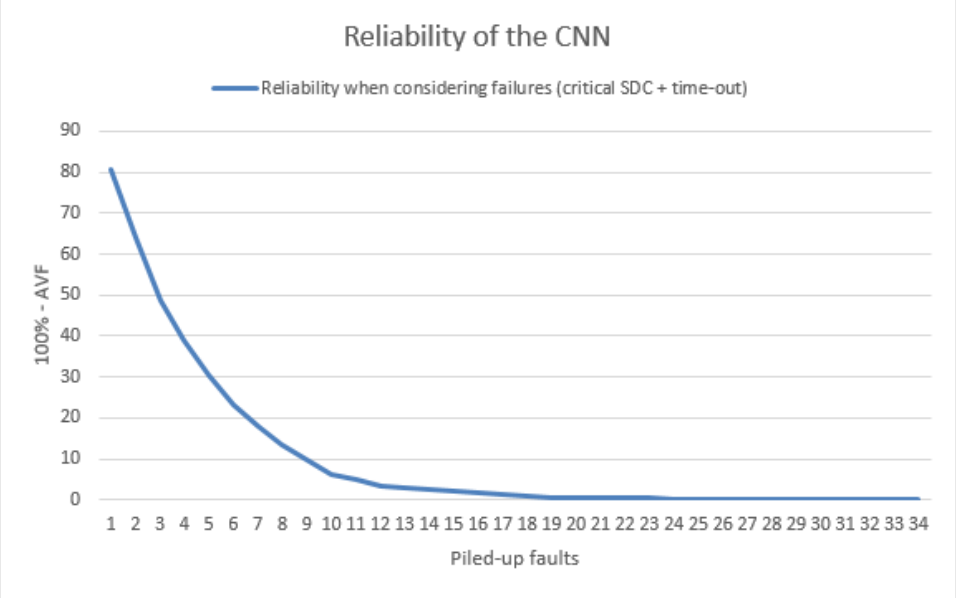
Piled-up faults	**1 image**	**2 images**	**3 images**	**4 images**	**5 images**	**6 images**	**Total**
1	68	46	24	18	23	16	195
2	135	84	40	29	32	35	355
3	197	118	52	46	52	43	508
4	236	139	62	56	60	59	612
5	271	156	74	62	65	64	692
6	301	169	78	72	76	70	766
7	325	178	84	78	81	74	820
8	345	186	88	83	85	76	863
9	360	192	94	87	88	81	902
10	372	196	95	90	94	90	937
11	380	200	95	90	94	90	949
12	386	202	97	90	96	95	966
13	388	202	98	91	96	95	970
14	390	203	98	91	96	95	973
15	394	203	98	91	97	96	979
16	396	204	99	91	97	96	983
17	397	205	99	91	97	97	986
18	399	205	100	91	97	98	990
19	401	205	100	91	98	99	994
20	401	205	100	91	98	99	994
21	402	205	100	91	98	99	995
22	403	205	100	91	98	99	996
23	403	205	100	91	98	99	996
24	404	205	100	91	98	99	997
25	404	205	100	91	99	99	998
26	404	205	100	91	99	99	998
27	404	205	100	91	99	99	998
28	404	205	100	91	99	99	998
29	404	205	100	91	99	99	998
30	404	205	100	91	99	99	998
31	404	205	100	91	99	99	998
32	404	205	100	91	99	100	999
33	404	205	100	91	99	100	999
34	405	205	100	91	99	100	1000

Figure 7.6: Number of wrong classifications



Source: The Author

Figure 7.7: Failure reliability



Source: The Author

(critical SDC + time-outs) is about 80.5% and exponentially decreases with the number of piled-up faults. The maximum number of piled-up faults before a failure occurs is 34 but this case has a minimum chance of happening. As the essential bits comprise 57.33% out of the entire physical block configuration bits and the AVF due to SEU is about 19.5%, the critical bits of the proposed design are approximately 34.03% (19.5/57.33) out of the essential bits.

Related work (LOPES; KASTENSMIDT; SUSIN, 2017)(CLEMENTE et al., 2016) shows that Neural Networks can have different AVFs depending on the topology of the Neural Network and its hardware architecture. In (LOPES; KASTENSMIDT; SUSIN, 2017) the Neural Network AVF is about 0.62% while the AVF of the proposed CNN is about 19.5%. Nevertheless, the Neural Networks implemented by (LOPES; KASTENSMIDT; SUSIN, 2017) do not use timing-multiplexing for area-saving, rather a full-parallel approach was implemented. In the proposed timing-multiplexing architecture, hardware resources are reused in many computations, hence an upset in a resource will affect many computations, increasing the probability of a fault being propagated to the output. In the full-parallel approach each one of the neuron inputs has dedicated signals, consequently, an upset in a resource will be associated to a specific signal and can be masked by the combination of other signals decreasing the probability of a fault being propagated to the output.

The AVF of the Hopfield Neural Network (HNN) implemented by (CLEMENTE et al., 2016) is about 15.64% (Errors, time-out and convergence), that is near the AVF of the proposed CNN, 19.5%. This Neural Network topology also reuses hardware resources, inasmuch as it has feedback and after a specific number of cycles the output converges to a value. An "error" occurs when the HNN converges to a wrong value, a "time-out" occurs when after a large number of cycles the HNN does not return any result, and a "convergence" occurs when the HNN takes more cycles than that expected to converge to the correct result. Provided that the "convergence" is the main portion of the AVF (13.28%), many differences in the output can be corrected by the feedback, but nonetheless it takes more cycles to converge. Rather, in the proposed CNN, the errors are masked by the softmax and max function computed at the ARM processor. Therefore, it is possible to note that Neural Networks that reuse hardware resources are more susceptible than parallel pipe-lined Neural Networks.

Furthermore, there is the possibility of a fault being injected in a configuration bit that transforms LUT used as memory into a shift-register. As mentioned previously,

in this case, a SBU in this specific configuration bit will cause multiple differences in other configuration bits (NAZAR; CARRO, 2012). As mentioned in section 5.3, LUTs as memory were extensively used to store the Full-connected weights, therefore there is a great probability that these LUTs will be affected by the random fault injection.

8 CONCLUSIONS

A CNN was proposed in this work to classify traffic-sign images from the GTSRB dataset. Due to the fact that one of the goals of this work was to obtain good average accuracy results in a short time, the training results of the super-class CNNs show a good average accuracy (90.0%) for a short time training (40 minutes) on a single GTX 680 GPU card. Another work (CIREŞAN et al., 2012) that achieves a state-of-the-art average accuracy (99.46%) took 37 hours to be trained on four GTX 580 GPU cards. Therefore, the proposed CNN training achieves an acceptable average accuracy result taken into account its training time. In virtue of the fact that there is no specific method for defining a neural network model and solver parameters, the trial-and-error method is used for neural network training (HAYKIN; NETWORK, 2004). Even if the tests are automatically performed, there are numerous possibilities. Thus, neural network training is a very time-consuming task, inasmuch as several layers, architectures, solvers, and even other machine learning approaches must be tested in order to achieve excellent results. Hence, if a more time-consuming CNN training was carried out, it would reach state-of-the-art accuracy results. Unfortunately, due to the available APSoC resources, the initially proposed CNN had to be reduced to a 6-sub-class CNN but its average accuracy result was maintained.

The second goal of this work was to accelerate the trained deep learning algorithm in hardware, thus, given the APSoC implementation results, the timing-multiplexing architecture obtained good performance results considering the available mid-range APSoC device. Given the performance scalability of the CNN design, if an APSoC device with more resources, like the Zynq Ultrascale family, is used, it is possible to implement the initially proposed CNN that classifies all the prohibitory traffic-signs maintaining practically the same performance. As the performance of the proposed CNN is better than or nearly equal to the related work (CIREŞAN et al., 2012)(WANG et al., 2013)(LIU et al., 2016)(YANG et al., 2016)(KASSANI; TEOH, 2017), the performance results justify the area utilized. Moreover, if a slower CNN is required, the CNN can be timing-multiplexed at more levels, saving even more area. Regarding the power results, as expected for an APSoC device, the cost of the power consumption is lower than GPUs, even on embedded GPUs (HUANG, 2016)(NVIDIA, 2016), decreasing the thermal dissipation of the system.

Lastly, the third goal of this work was to analyze the reliability of the proposed

hardware implementation. About the reliability of CNNs which were implemented on FPGAs or PL part of APSoC, comparing the AVF of the proposed CNN (19.05%) and the HNN (15.64%) implemented by (CLEMENTE et al., 2016) with another work that analyzes neural network (0.62%) (LOPES; KASTENSMIDT; SUSIN, 2017), it was possible to note that the area-saving associated to the timing multiplexed-architecture of the proposed CNN and the former cited HNN considerably increase the AVF, inasmuch as the full-parallel architecture of the latter cited work presented an AVF much smaller. Hence, full-parallel neural networks implemented on FPGAs/APSoC add redundancy to the design with the cost of high area-overhead. In order to make a radiation reliable traffic-sign recognition system, the implementation must be verified by fault injection or a radiation experiment. Then, a redundancy approach which can balance the area, power, performance, and reliability has to be chosen, like Approximate Triple Modular Redundancy (TMR) (GOMES; KASTENSMIDT, 2013).

8.1 Future Work

In order to infer which approach has high reliability and performance, and low area and power costs, other machine learning algorithms like Random Forests and SVMs, can be explored. Nevertheless, in order to reduce the design time required for accelerating these algorithms in hardware, other VHDL automatic generator for these approaches must be developed, and then making their tests less timing consuming. For more embracing reliability results, the whole traffic-sign system encompassing the detection and classification phase must be implemented. Moreover, even other autonomous car tasks such as pedestrian detection, automobile detection, road detection and lane detection can be explored. In addition, other components of the APSoC device can be tested by radiation experiments such as the ARM processor, the BRAM and DDR memories. Additionally, other fault injection by emulation approaches can be tested such as fault injection in the combinational logic (CLBs) or sequential logic (Flip-Flops).

REFERENCES

ALDERIGHI, M. et al. Evaluation of single event upset mitigation schemes for sram based fpgas using the flipper fault injection platform. In: IEEE. **Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on**. [S.l.], 2007. p. 105–113.

ALVES, T. W. **Sistema de detecção em tempo real de faixas de sinalização de trânsito para veículos inteligentes utilizando processamento de imagem**. 2017. Disponível em: <http://hdl.handle.net/10183/157872>. Acesso em: 21 ago. 2017.

BAUMANN, R. et al. Boron compounds as a dominant source of alpha particles in semiconductor devices. In: IEEE. **Reliability Physics Symposium, 1995. 33rd Annual Proceedings., IEEE International**. [S.l.], 1995. p. 297–302.

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

BERGER, A. S. **Embedded systems design: an introduction to processes, tools, and techniques**. [S.l.]: Focal Press, 2002.

BERNARDI, P. et al. On the evaluation of seu sensitiveness in sram-based fpgas. In: IEEE. **On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings. 10th IEEE International**. [S.l.], 2004. p. 115–120.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV: Computer vision with the OpenCV library**. [S.l.]: " O'Reilly Media, Inc.", 2008.

BRIDLE, J. S. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In: **Neurocomputing**. [S.l.]: Springer, 1990. p. 227–236.

CAFFE. **Caffe Solving in Python with LeNet**. 2017. Available at: <http://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/01-learning-lenet.ipynb>. Accessed Aug. 25, 2017.

CIREŞAN, D. et al. Multi-column deep neural network for traffic sign classification. **Neural Networks**, Elsevier, v. 32, p. 333–338, 2012.

CLEMENTE, J. A. et al. Hardware implementation of a fault-tolerant hopfield neural network on fpgas. **Neurocomputing**, Elsevier, v. 171, p. 1606–1609, 2016.

DERRY, T. K.; WILLIAMS, T. I. **A short history of technology from the earliest times to AD 1900**. [S.l.]: Courier Corporation, 1960. v. 231.

DETTMERS, T. 8-bit approximations for parallelism in deep learning. **arXiv preprint arXiv:1511.04561**, 2015.

DING, C. et al. Matrix multiplication on gpus with on-line fault tolerance. In: IEEE. **Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on**. [S.l.], 2011. p. 311–317.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, n. Jul, p. 2121–2159, 2011.

FU, Y. et al. **Xilinx Deep Learning with INT8 Optimization on Xilinx Devices**. 2017. Available at: <https://www.xilinx.com/support/documentation/white_papers/wp486-deep-learning-int8.pdf>. Accessed Aug. 25, 2017.

FUENTES-P, J.; RUIZ-A, J.; RENDÓN-M, J. M. Visual simultaneous localization and mapping: a survey. **Artificial Intelligence Review**, Springer, v. 43, n. 1, p. 55–81, 2015.

GADLAGE, M. J. et al. Scaling trends in set pulse widths in sub-100 nm bulk cmos processes. **IEEE Transactions on Nuclear Science**, IEEE, v. 57, n. 6, p. 3336–3341, 2010.

GERLA, M. et al. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In: IEEE. **Internet of Things (WF-IoT), 2014 IEEE World Forum on**. [S.l.], 2014. p. 241–246.

GERONIMO, D. et al. Survey of pedestrian detection for advanced driver assistance systems. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 32, n. 7, p. 1239–1258, 2010.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2014. p. 580–587.

GOMES, I. A.; KASTENSMIDT, F. G. Reducing tmr overhead by combining approximate circuit, transistor topology and input permutation approaches. In: IEEE. **Integrated Circuits and Systems Design (SBCCI), 2013 26th Symposium on**. [S.l.], 2013. p. 1–6.

GONZALEZ, R. C.; WOODS, R. E. Digital image processing prentice hall. **Upper Saddle River, NJ**, 2002.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GYSEL, P.; MOTAMEDI, M.; GHIASI, S. Hardware-oriented approximation of convolutional neural networks. **arXiv preprint arXiv:1604.03168**, 2016.

HAN, Y.; ORUKLU, E. Real-time traffic sign recognition based on zynq fpga and arm socs. In: IEEE. **Electro/Information Technology (EIT), 2014 IEEE International Conference on**. [S.l.], 2014. p. 373–376.

HAYKIN, S. **Neural networks: a comprehensive foundation**. [S.l.]: Prentice Hall PTR, 1994.

HAYKIN, S.; NETWORK, N. A comprehensive foundation. **Neural Networks**, v. 2, n. 2004, p. 41, 2004.

HE, K. et al. Spatial pyramid pooling in deep convolutional networks for visual recognition. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2014. p. 346–361.

HOELSCHER, I. G. **Detecção e classificação de sinalização vertical de trânsito em cenários complexos**. 2017. Disponível em: <http://hdl.handle.net/10183/163777>. Acesso em: 21 ago. 2017.

HOHL, J. H.; GALLOWAY, K. F. Analytical model for single event burnout of power mosfets. **IEEE Transactions on Nuclear Science**, IEEE, v. 34, n. 6, p. 1275–1280, 1987.

HOUBEN, S. et al. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In: **International Joint Conference on Neural Networks**. [S.l.: s.n.], 2013.

HSUEH, M.-C.; TSAI, T. K.; IYER, R. K. Fault injection techniques and tools. **Computer**, IEEE, v. 30, n. 4, p. 75–82, 1997.

HUANG, J.-H. **NVidia ACCELERATING THE RACE TO SELF-DRIVING CARS**. 2016. Available at: <http://nvidianews.nvidia.com/_ir/219/20160/JHH_CES2016_FINAL_published.pdf>. Accessed Aug. 25, 2017.

ISO-26262. **ISO 26262-1:2011 Preview Road vehicles – Functional safety – Part 1: Vocabulary**. 2011. Available at: <https://www.iso.org/standard/43464.html>. Accessed Aug. 23, 2017.

ISON1570. **ISO/IEC 9899:201x Committee Draft April 12, 2011 N1570**. 2011. Available at: <http://iso-9899.info/n1570.html>. Accessed Aug. 23, 2017.

JAIN, A. K.; MAO, J.; MOHIUDDIN, K. M. Artificial neural networks: A tutorial. **Computer**, IEEE, v. 29, n. 3, p. 31–44, 1996.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. **arXiv preprint arXiv:1408.5093**, 2014.

KAJIHARA, S. et al. Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 14, n. 12, p. 1496–1504, 1995.

KASSANI, P. H.; TEOH, A. B. J. A new sparse model for traffic sign classification using soft histogram of oriented gradients. **Applied Soft Computing**, Elsevier, v. 52, p. 231–246, 2017.

KASTENSMIDT, F. L.; CARRO, L.; REIS, R. A. da L. **Fault-tolerance techniques for SRAM-based FPGAs**. [S.l.]: Springer, 2006. v. 32.

KOGA, R. et al. Single event functional interrupt (sefi) sensitivity in microcircuits. In: IEEE. **Radiation and Its Effects on Components and Systems, 1997. RADECS 97. Fourth European Conference on**. [S.l.], 1997. p. 311–318.

LE, R. **Xilinx Soft Error Mitigation Using Prioritized Essential Bits**. 2012. Available at: <https://www.xilinx.com/support/documentation/application_notes/xapp538-soft-error-mitigation-essential-bits.pdf>. Accessed Aug. 24, 2017.

LECUN, Y.; BENGIO, Y. et al. Convolutional networks for images, speech, and time series. **The handbook of brain theory and neural networks**, v. 3361, n. 10, p. 1995, 1995.

LEE, K. J. et al. A 502-gops and 0.984-mw dual-mode intelligent adas soc with real-time semiglobal matching and intention prediction for smart automotive black box system. **IEEE Journal of Solid-State Circuits**, IEEE, v. 52, n. 1, p. 139–150, 2017.

LI, X. et al. Online estimation of architectural vulnerability factor for soft errors. In: IEEE. **Computer Architecture, 2008. ISCA'08. 35th International Symposium on**. [S.l.], 2008. p. 341–352.

LIU, C. et al. Fast traffic sign recognition via high-contrast region extraction and extended sparse representation. **IEEE transactions on Intelligent transportation systems**, IEEE, v. 17, n. 1, p. 79–92, 2016.

LIU, F.; NARAYANAN, A.; BAI, Q. Real-time systems. Citeseer, 2000.

LOPES, I. C.; KASTENSMIDT, F. L.; SUSIN, A. A. Seu susceptibility analysis of a feedforward neural network implemented in a sram-based fpga. In: IEEE. **Test Symposium (LATS), 2017 18th IEEE Latin American**. [S.l.], 2017. p. 1–6.

MUKHERJEE, S. S. et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: IEEE. **Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on**. [S.l.], 2003. p. 29–40.

MUSSEAU, O. et al. Analysis of multiple bit upsets (mbu) in cmos sram. **IEEE Transactions on Nuclear Science**, IEEE, v. 43, n. 6, p. 2879–2888, 1996.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: **Proceedings of the 27th international conference on machine learning (ICML-10)**. [S.l.: s.n.], 2010. p. 807–814.

NAZAR, G. L. Fine-grained error detection techniques for fast repair of fpgas. 2013.

NAZAR, G. L.; CARRO, L. Fast single-fpga fault injection platform. In: IEEE. **Defect and fault tolerance in VLSI and nanotechnology systems (DFT), 2012 IEEE international symposium on**. [S.l.], 2012. p. 152–157.

NHTSA. **NHTSA TRAFFIC SAFETY FACTS**. 2013. Available at: <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/811856>. Accessed Aug. 25, 2017.

NVIDIA. **Nvidia The AI Car Computer for Autonomous Driving**. 2016. Available at: <http://www.nvidia.com/object/drive-px.html>. Accessed Aug. 25, 2017.

NVIDIA. **Nvidia Graphics Processing Unit (GPU)**. 2017. Available at: <http://www.nvidia.com/object/gpu.html>. Accessed Aug. 25, 2017.

O'GORMAN, T. J. et al. Field testing for cosmic ray soft errors in semiconductor memories. **IBM Journal of Research and Development**, IBM, v. 40, n. 1, p. 41–50, 1996.

OHLSSON, M. et al. Neutron single event upsets in sram-based fpgas. In: IEEE. **Radiation Effects Data Workshop, 1998. IEEE**. [S.l.], 1998. p. 177–180.

OLDHAM, T. R.; MCLEAN, F. Total ionizing dose effects in mos oxides and devices. **IEEE Transactions on Nuclear Science**, IEEE, v. 50, n. 3, p. 483–499, 2003.

POUGET, V. et al. Theoretical investigation of an equivalent laser let. **Microelectronics Reliability**, Elsevier, v. 41, n. 9-10, p. 1513–1518, 2001.

RABAEY, J. M.; CHANDRAKASAN, A. P.; NIKOLIC, B. **Digital integrated circuits**. [S.l.]: Prentice hall Englewood Cliffs, 2002. v. 2.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2016. p. 779–788.

REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2015. p. 91–99.

SANTOS, F. F. d. **Reliability evaluation and error mitigation in pedestrian detection algorithms for embedded GPUs**. 2017. Available at: <http://hdl.handle.net/10183/159210>. Accessed Aug. 21, 2017.

SEDRA, A. S.; SMITH, K. C. **Microelectronic circuits**. [S.l.]: New York: Oxford University Press, 1998. v. 1.

SERMANET, P.; LECUN, Y. Traffic sign recognition with multi-scale convolutional networks. In: IEEE. **Neural Networks (IJCNN), The 2011 International Joint Conference on**. [S.l.], 2011. p. 2809–2813.

SINHA, P. Architectural design and reliability analysis of a fail-operational brake-by-wire system from iso 26262 perspectives. **Reliability Engineering & System Safety**, Elsevier, v. 96, n. 10, p. 1349–1359, 2011.

STALLKAMP, J. et al. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In: **IEEE International Joint Conference on Neural Networks**. [S.l.: s.n.], 2011. p. 1453–1460.

STALLKAMP, J. et al. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. **Neural Networks**, n. 0, p. –, 2012. ISSN 0893-6080. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0893608012000457>.

STEPHEN, J. et al. Cosmic ray simulation experiments for the study of single event upsets and latch-up in cmos memories. **IEEE Transactions on Nuclear Science**, IEEE, v. 30, n. 6, p. 4464–4469, 1983.

STRIGL, D.; KOFLER, K.; PODLIPNIG, S. Performance and scalability of gpu-based convolutional neural networks. In: IEEE. **Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on**. [S.l.], 2010. p. 317–324.

VELAZCO, R. et al. Evidences of seu tolerance for digital implementations of artificial neural networks: one year mptb flight results. In: IEEE. **Radiation and Its Effects on Components and Systems, 1999. RADECS 99. 1999 Fifth European Conference on**. [S.l.], 1999. p. 565–568.

VISWANATHAN, V.; HUSSEIN, R. Applications of image processing and real-time embedded systems in autonomous cars: A short review. **International Journal of Image Processing (IJIP)**, v. 11, n. 2, p. 35, 2017.

WANG, G. et al. A hierarchical method for traffic sign classification with support vector machines. In: IEEE. **Neural Networks (IJCNN), The 2013 International Joint Conference on**. [S.l.], 2013. p. 1–6.

WANG, W. et al. An efficient method to identify critical gates under circuit aging. In: IEEE. **Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on**. [S.l.], 2007. p. 735–740.

WESTE, N.; HARRIS, D.; BANERJEE, A. Cmos vlsi design. **A circuits and systems perspective**, v. 11, p. 739, 2005.

WIRTHLIN, M. et al. The reliability of fpga circuit designs in the presence of radiation induced configuration upsets. In: IEEE. **Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on**. [S.l.], 2003. p. 133–142.

XILINX. **Xilinx 7 Series DSP48E1 Slice**. 2016. Available at: <https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf>. Accessed 23 Aug. 23, 2017.

XILINX-AUTOMOTIVE. **Smarter Vision: Intelligence for Advanced Driver Assistance Systems**. 2016. Available at: <https://www.xilinx.com/applications/automotive.html>. Accessed Aug. 21, 2017.

XILINX-ZYNQ. **Zynq-7000 All Programmable SoC Product Advantages**. 2017. Available at: <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>. Accessed Aug. 25, 2017.

YANG, Y. et al. Towards real-time traffic sign detection and classification. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 17, n. 7, p. 2022–2031, 2016.

ZIADE, H. et al. A survey on fault injection techniques. **Int. Arab J. Inf. Technol.**, v. 1, n. 2, p. 171–186, 2004.

ZIEGLER, J. F. et al. Accelerated testing for cosmic soft-error rate. **IBM Journal of Research and Development**, IBM, v. 40, n. 1, p. 51–72, 1996.

ZUIDERVELD, K. Contrast limited adaptive histogram equalization. In: ACADEMIC PRESS PROFESSIONAL, INC. **Graphics gems IV**. [S.l.], 1994. p. 474–485.