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

FERNANDO FERNANDES DOS SANTOS

Reliability evaluation and error mitigation in pedestrian detection algorithms for embedded GPUs

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Paolo Rech

Porto Alegre
May 2017

CIP — CATALOGING-IN-PUBLICATION

Santos, Fernando Fernandes dos

Reliability evaluation and error mitigation in pedestrian detection algorithms for embedded GPUs / Fernando Fernandes dos Santos. – Porto Alegre: PPGC da UFRGS, 2017.

73 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2017. Advisor: Paolo Rech.

1. Fault tolerance. 2. Soft error. 3. Pedestrian detection. 4. Hardening. I. Rech, Paolo. II. Título.

“Education is an important element in the struggle for human rights.
It is the means to help our children and our people rediscover their identity and
thereby increase their self respect. Education is our passport to the future,
for tomorrow belongs only to the people who prepare for it today.

— Sir Malcolm X

ABSTRACT

Pedestrian detection reliability is a fundamental problem for autonomous or aided driving. Methods that use object detection algorithms such as Histogram of Oriented Gradients (HOG) or Convolutional Neural Networks (CNN) are today very popular in automotive applications. Embedded Graphics Processing Units (GPUs) are exploited to make object detection in a very efficient manner. Unfortunately, GPUs architecture has been shown to be particularly vulnerable to radiation-induced failures. This work presents an experimental evaluation and analytical study of the reliability of two types of object detection algorithms: HOG and CNNs. This research aim is not just to quantify but also to qualify the radiation-induced errors on object detection applications executed in embedded GPUs. HOG experimental results were obtained using two different architectures of embedded GPUs (Tegra and AMD APU), each exposed for about 100 hours to a controlled neutron beam at Los Alamos National Lab (LANL). Precision and Recall metrics are considered to evaluate the error criticality. The reported analysis shows that, while being intrinsically resilient (65% to 85% of output errors only slightly impact detection), HOG experienced some particularly critical errors that could result in undetected pedestrians or unnecessary vehicle stops. This works also evaluates the reliability of two Convolutional Neural Networks for object detection: You Only Look Once (YOLO) and Faster RCNN. Three different GPU architectures were exposed to controlled neutron beams (Kepler, Maxwell, and Pascal) detecting objects in both Caltech and Visual Object Classes data sets. By analyzing the neural network corrupted output, it is possible to distinguish between tolerable errors and critical errors, i.e., errors that could impact detection. Additionally, extensive GDB-level and architectural-level fault-injection campaigns were performed to identify HOG and YOLO critical procedures. Results show that not all stages of object detection algorithms are critical to the final classification reliability. Thanks to the fault injection analysis it is possible to identify HOG and Darknet portions that, if hardened, are more likely to increase reliability without introducing unnecessary overhead. The proposed HOG hardening strategy is able to detect up to 70% of errors with a 12% execution time overhead.

Keywords: Fault tolerance. soft error. pedestrian detection. hardening.

Validação da confiabilidade e tolerância a falhas em algoritmos de detecção de pedestres para GPUs embarcadas

RESUMO

A confiabilidade de algoritmos para detecção de pedestres é um problema fundamental para carros auto dirigíveis ou com auxílio de direção. Métodos que utilizam algoritmos de detecção de objetos como Histograma de Gradientes Orientados (HOG - *Histogram of Oriented Gradients*) ou Redes Neurais de Convolução (CNN – *Convolutional Neural Network*) são muito populares em aplicações automotivas. Unidades de Processamento Gráfico (GPU – *Graphics Processing Unit*) são exploradas para executar detecção de objetos de uma maneira eficiente. Infelizmente, as arquiteturas das atuais GPUs tem se mostrado particularmente vulneráveis a erros induzidos por radiação. Este trabalho apresenta uma validação e um estudo analítico sobre a confiabilidade de duas classes de algoritmos de detecção de objetos, HOG e CNN. Esta pesquisa almeja não somente quantificar, mas também qualificar os erros produzidos por radiação em aplicações de detecção de objetos em GPUs embarcadas. Os resultados experimentais com HOG foram obtidos usando duas arquiteturas de GPU embarcadas diferentes (Tegra e AMD APU), cada uma foi exposta por aproximadamente 100 horas em um feixe de nêutrons em *Los Alamos National Lab* (LANL). As métricas *Precision* e *Recall* foram usadas para validar a criticalidade do erro. Uma análise final mostrou que por um lado HOG é intrinsecamente resiliente a falhas (65% a 85% dos erros na saída tiveram um pequeno impacto na detecção), do outro lado alguns erros críticos aconteceram, tais que poderiam resultar em pedestres não detectados ou paradas desnecessárias do veículo. Este trabalho também avaliou a confiabilidade de duas Redes Neurais de Convolução para detecção de Objetos:Darknet e Faster RCNN. Três arquiteturas diferentes de GPUs foram expostas em um feixe de nêutrons controlado (Kepler, Maxwell, e Pascal), com as redes detectando objetos em dois data sets, Caltech e *Visual Object Classes*. Através da análise das saídas corrompidas das redes neurais, foi possível distinguir entre erros toleráveis e erros críticos, ou seja, erros que poderiam impactar na detecção de objetos. Adicionalmente, extensivas injeções de falhas no nível da aplicação (GDB) e em nível arquitetural (SASSIFI) foram feitas, para identificar partes críticas do código para o HOG e as CNNs. Os resultados mostraram que não são todos os estágios da detecção de objetos que são críticos para a confiabilidade da detecção final. Graças a injeção de falhas foi possível identificar partes do HOG e da Darknet, que se protegidas, irão com uma maior probabilidade aumentar a sua confiabilidade, sem adicionar um *overhead* desnecessário. A estratégia de tolerância a falhas proposta para o HOG foi capaz de detectar até 70% dos erros com 12% de *overhead* de tempo.

Palavras-chave: tolerância a falhas,soft error, detecção de pedestres,proteção por software.

LIST OF ABBREVIATIONS AND ACRONYMS

APU	Accelerated Processing Unit
ADAS	Advanced Driver Assistance Systems
AVF	Architectural Vulnerability Factor
ANN	Artificial Neural Network
ASIL	Automotive Safety Integrity Level
BB	Bouding Box
COTS	Commercial off-the-shelf
CUDA	Compute Unified Device
CNN	Convolutional Neural Network
cuDNN	cuda Deep Neural Networks library
ECC	Error Correction Code
Euro NCAP	European New Car Assessment
FIT	Failure In Time
FTTI	Fault Tolerant Time Interval
FLOPS	Float Operations Per Second
GEMM	General Matrix Multiplication
GDB	GNU project Debugger
GPU	Graphics Processing Unit
HPC	High Performance Computer
HOG	Histogram of Oriented Gradients
INST	Instruction injection site
MPH	Miles Per Hour
NHTSA	National Highway Traffic Safety Administration
PVF	Program Vulnerability Factor

RoI	Region of Interest
RPN	Region Proposal Network
RCNN	Region-based Convolutional Neural Network
RF	Register File injection site
RTL	Register Transfer Level
SASSIFI	SASSI Based Fault Injector
SDC	Silent Data Corruption
SIMD	Single Instruction Multiple Data]
SER	Soft Error Rate
SVM	Support Vector Machine
SoC	System on Chip
YOLO	You Only Look Once

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Embedded systems are nowadays common used in applications that range from a simple smart watch to a complex self-driving car system. A system, to be called embedded, processes some data from the environment and, based on it, generates control signals to manage the system they are integrated in (YEN; WOLF, 2013). Since the embedded system must follow specific and, at times, strict constraints, designing and programming an embedded system could be much more challenging than programming a general purpose computing device.

In the last years, the embedded systems market have been demanding much more processing and power performance. The new market scenario is forcing engineers to put their effort in produces high-performance embedded systems, which must consume less power than a general purpose computer, in some cases must be real time (WOLF, 2012; WOLF, 2014).

According to Marwedel Embedded systems must fit at leat these three specific constraints Dependability, Energy Efficiency, and Performance (MARWEDEL, 2010): (1) Dependability is essential especially for those applications which the device must be reliable and maintainable. An embedded system is typically required to maintain its functionality for a given period. In particular, safety-critical applications require very high reliability, i.e., applications where an error or malfunction could cause life loss or several damages. (2) Energy efficiency is nowadays of extreme importance, specifically for those applications for which there is a limited power/energy budget, like portable devices. It is then likely to have an embedded system design that consumes less power/energy as possible. (3) Finally, performance is a critical constraint on a real-time system, since a delayed answer could result in a user harm or property loss.

In some specific situations, embedded systems are required to respect not just one or few, but all the mentioned requirements. Self-driving cars, aircraft, military, and space applications are examples in which the system must be reliable while manipulating a significant amount of data in real-time and be extremely energy efficient. In this scenario, designing an embedded system which fills all requirements is an extremely challenging task.

Recently embedded GPUs have got into the market, bringing the processing power which was reserved for HPC research into embedded systems. Thanks to their low cost, increased energy efficiency, and flexible development platforms, low-power embedded

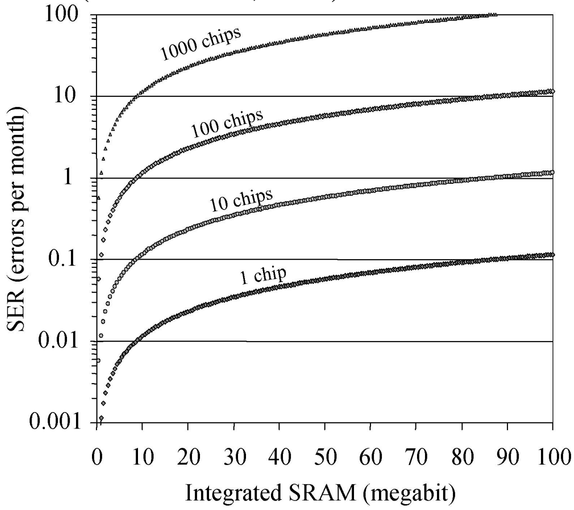
GPUs have enjoyed widespread adoption in various application domains. The employment of embedded GPUs on the embedded systems is a reality. Tesla self-driving system, for instance, is powered by NVIDIA Tegra K1 and X1 embedded GPUs. However, Tesla is not the only enterprise that is working with NVIDIA, others such as Audi, Mercedes-Benz, Volvo, Baidu, etc.. All of then want to exploit embedded GPUs to enter the self-driven car's market (NVIDIA, 2017).

Efficient parallel processing is attractive to compress images in satellites and reduce the bandwidth necessary to send them to ground (AGENCY, 2014). On aircraft, GPUs are studied to integrate all the circuitry required to implement collision avoidance systems (BECKER; SANDER, 2013). An application is considered safety critical if it should not fail, or if a failure could result in a loss of life, property damage, or environment damage (KNIGHT, 2002; BLANCHET et al., 2003). Some embedded systems are used in safety critical system, so they have to be reliable, which makes their development process harder.

With the shrink of transistors size to attend the market demand for low power consumption and high performances, a new reliability problem for electronic devices arises. Transistors become more and more susceptible to single events caused by radiation particles (BAUMANN, 2005). When a radiation particle has enough energy to change the transistor state, it may change values of registers, flip-flops, memory cells, or latches. Additionally, radiation can also disturb logic computation. Terrestrial radiation-induced errors are mainly $soft$ errors, once they do not cause permanent damage to the circuit. When an application uses the corrupted data it could process a variable in a wrong way, resulting in a corrupted software output, called as Silent Data Corruption (SDC).

The probability of a single transistor to be affected by a soft error is small, that is, the single transistor Soft Errors Rate (SER) is low. However, Baumann has already shown that if electronic chips are massively used in parallel, the SER will increase significantly (BAUMANN, 2005). Figure 1.1 shows the monthly SER for the amount of SRAM memory integrated on the device, it indicates that when lots of transistors are used in parallel, the system SER becomes a critical issue for electronic system projects. More recent works have been shown that SER is also a huge concern on GPUs (OLIVEIRA et al., 2014b; PILLA et al., 2015; OLIVEIRA et al., 2016). Unfortunately, as GPUs were traditionally designed for entertaining and gaming applications, for which soft error rate is not critical, their reliability solutions are in its beginning if compared with general purpose CPUs.

Figure 1.1: Monthly Soft Error Rate SER scale per number of chips. The image shows for one memory chip the SER will be negligible, but when 1000 chips are used in parallel it can not be ignored. From (BAUMANN, 2005).



1.1 Motivation

The terrestrial atmosphere is constantly bathed by cosmic rays, which are cumbersome and energetic charged particles from space (figure 1.2). Most of the cosmic rays are captured or blocked by the Earth's magnet field (GOLDHAGEN, 2003), however, depending on the particle energy, it could penetrate and hit the upper level of our atmosphere, producing a significant amount of neutrons that reach the ground. Energetic particles are a real concern on electronic devices reliability. In the past, lots of work have been reporting problems with cosmic rays on electronic devices, not only on satellites (BINDER; SMITH; HOLMAN, 1975) and airplanes (DICELLO; PACIOTTI; SCHILLACI, 1989), but also at the sea level (MAY; WOODS, 1979; MICHALAK et al., 2012).

A soft error could result in an unexpected application behavior, which may lead to some environment damage, a risk of a human life or even death. A radiation-induced event on an electronic system could cause various unexpected results, such as arithmetic errors, modifications on reconfigurable logic circuits, a wrong processing output, operating system crash, and other fails.

Soft Errors are particularly critical for automotive safety-critical systems as a failure caused by an SDC could bring serious damage. The most famous case is Jean Bookout Vs. Toyota Motors (TIMES, 2013). After a technical problem a Toyota Camry 2005 car with two women inside, suddenly accelerated until it crashed, leaving one woman dead and Jean Bookout seriously injured (figure 1.3). During the trial, a reverse engineering process demonstrates that two things culminated to the accident: (1) bad programming

Figure 1.2: Cosmmic ray incidence on earth. From (NASA, 2005).

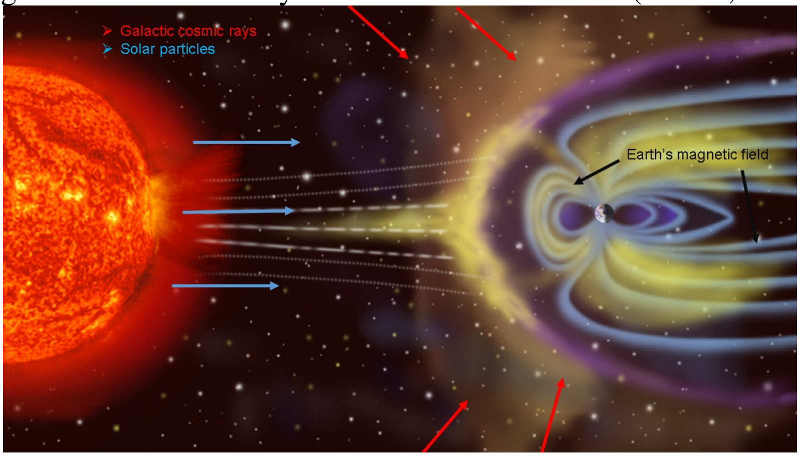


Figure 1.3: Toyota Camry 2005 after the crash in a highway. From (MONEY, 2013).

practices and (2) a bit flip that killed a system task (KOOPMAN, 2014; CUMMINGS, 2016; STRATEGIES, 2013). Toyota Motors was sentenced to pay $ 1.5 million dollars to Jean Bookout and $ 1.5 million to the other woman family.

The accident mentioned above helps to understand the importance of reliability on automotive embedded systems. Car enterprises want to put more embedded applications on their products, so it is necessary to ensure that all safety metrics will be met. Autonomous cars are complex by its definition, as it is a combination of many Artificial Intelligence algorithms, such as Object Detection, Free Space Pixel Labeling, Sign Detection, and others AI methods. Putting all cited AI techniques on a final product and guaranteeing safety is extremely challenging. In this scenario, this work will contribute to evaluate the reliability of three Object Detection algorithms, Histogram of Oriented Gradients, and two different Convolutional Neural Networks.

1.2 Goals and Contributions

The main contributions of this work are:

1. The experimental evaluation of the behavior of HOG executed on embedded GPUs exposed to radiation;
2. Experimental evaluation of two Convolutional Neural Networks (Darknet and Faster RCNN) under radiation, using the most advanced GPU architectures available on the market;
3. Utilization of formal metrics to describe the radiation-induced errors on pedestrian detection systems and to evaluate their criticality;
4. Identification of the most vulnerable HOG and Darknet procedures through fault-injection;
5. Concrete proposals on how to increase HOG and Darknet reliability for automotive applications;
6. Validation of HOG hardening strategies through SASSIFI fault injection.

The remainder of the document is organized as follows. Chapter 2 gives a background on reliability of electronic devices and functional safety for automotive applications concepts. Chapter 3 introduces the used object detection algorithms and their structure and reviews. Chapter 4 presents the proposed error criticality metrics, the radiation tests setup and the fault injection tools that are used in this work. Chapter 5 presents and discusses experimental and fault-injection results, while Chapter 6 concludes the document and presents some future works.

2 RADIATION EFFECTS ON ELECTRONIC CIRCUITS

This chapter will explain the main radiation effects on electronic circuits. Due to the transistors shrinking, it is necessary to study the impact of radiation-induced errors on embedded devices. Since GPUs are becoming popular on embedded systems, it is also mandatory to comprehend the implication of radiation-induced events on GPUs.

2.1 Reliability issue for electronics devices

Various sources of errors could undermine the system reliability, including environmental perturbations, software errors, temperature, or voltage variations (LUTZ, 1993; LAPRIE, 1995; NICOLAIDIS, 1999; BAUMANN, 2005). The generated error may corrupt data values or logic operations and lead to Silent Data Corruption (SDC), Application Crash, System Hang, or be masked and cause no observable error (CONSTANTINESCU, 2002; SAGGESE et al., 2005; SCHROEDER; PINHEIRO; WEBER, 2009a).

This work focuses on radiation-induced soft errors that, according to (BAUMANN, 2005), are a huge concern in modern computing devices because, uncorrected, they produce a failure rate that is higher than all the other reliability mechanisms combined.

In fact, today the radiation-induced failure rate represents a significant issue not only in radiation-harsh environments, such as space but also in milder environments at sea level. Due to the shrinking of transistor dimensions and the exacerbation of many available resources, electronic devices are becoming more susceptible to soft errors induced by ionizing particles. High-energy neutrons generated by the interaction of cosmic rays with the terrestrial atmosphere may then have enough energy to corrupt data stored in SRAM memories or to affect logic computations (BAUMANN, 2005). Radiation-induced failures are expressly pertinent in safety-critical applications, for which reliability is mandatory.

2.2 GPU Reliability

While extremely efficient in terms of *FLOP/s* and *FLOPs-per-WATT*, modern GPUs have been shown to be prone to experience radiation-induced corruption (DEBARDELEBEN et al., 2013; WUNDERLICH; BRAUN; HALDER, 2013; GOMEZ et al., 2014; OLIVEIRA et al., 2014a; OLIVEIRA et al., 2016). GPU architecture may be particularly susceptible to be corrupted by radiation for three main reasons. (1) GPUs were traditionally designed for entertainment video editing or gaming applications, for which reliability is not a concern (BREUER; GUPTA; MAK, 2004). Thus, their architecture is optimized to increase performance more than reliability. (2) GPUs orchestrate parallel processes using hardware scheduler and dispatcher. To make parallel executions more efficient, GPUs use shared memory among threads. The corruption of the scheduler or an error in the shared memory is likely to affect the computation of several parallel processes, significantly lowering the GPU reliability (RECH et al., 2013b). (3) GPUs require a huge amount of memory to implement parallelism, both caches and register files. These memories have been demonstrated to be the cause of the majority of errors in modern computing devices (SEIFERT; ZHU; MASSENGILL, 2002; TAN et al., 2011).

Typically CPU cores include parity, Error Correcting Code (ECC) or other error protection mechanisms on their main memory structures even in commercial SoCs (AMD, 2015; ARM, 2016; INTEL, 2016), most embedded GPUs, including the ones considered in this paper, still lack any protection. As an example, Tesla's self-diving system is powered by NVIDIA *Tegra K1* and *Tegra X1*, which do not have any reliability solution applied to the GPU core. The lack of architectural hardening makes embedded GPUs unprotected against soft errors, which could undermine their utilization for safety-critical systems. It is worth noting that the latest GPUs for High-Performance Computing Applications include ECC on the main memory structures (NVIDIA, 2015b; INTEL, 2014), which has been shown to reduce of about one order of magnitude the GPU error rate (OLIVEIRA et al., 2016). Lately, some SoCs with ECC-protected GPU memories appeared in the market (AMD, 2015) but are not employed in any self-driving car, yet. As discussed in Chapter 5, ECC is likely to improve SoCs reliability significantly and reduce software hardening overhead. However, ECC alone is still not sufficient to guarantee high reliability, as radiation can corrupt logic as well (BAUMANN, 2005).

Many studies were done on large-scale systems reliability(MARTINO et al., 2014; EL-SAYED; SCHROEDER, 2013; LIANG et al., 2006; LIANG et al., 2005; OLINER; STEARLEY, 2007; PECCHIA et al., 2011; SAHOO et al., 2004; SCHROEDER; GIBSON, 2010). Generally, large-scale systems reliability studies focus on DRAM (HWANG; STEFANOVICI; SCHROEDER, 2012; SCHROEDER; PINHEIRO; WEBER, 2009b; SRIDHARAN et al., 2013) and disk failures (SCHROEDER; GIBSON, 2007). Blue Waters system (MARTINO et al., 2014), primarily focusing on characterizing all system failures, reports only the number of uncorrectable errors on GPUs. This work is the first survey focused specifically on GPU failures caused by soft errors in embedded systems.

Driven by the proliferation of GPUs in HPC and safety-critical applications, many studies have been trying to improve GPU reliability (GOMEZ et al., 2014). GPUs reliability exposed on terrestrial radiation is also studied in some initial works (DEBARDELEBEN et al., 2013; RECH et al., 2014). At the same time, fault injection experiments have been performed to track error propagation towards the GPU outputs and evaluate the Architectural Vulnerability Factor (AVF) of several parallel codes (FANG et al., 2014a; FANG et al., 2012; HAQUE; PANDE, 2010; TAN et al., 2011)

This is the first study that provides radiation experiments for a variety of safety-critical workloads and evaluates the efficiency of ECC for critical applications. No prior work has combined the field study and radiation tests to understand the GPU failures in a unified way. This study provides insights into differences in the reliability of object detection applications for automotive area. The fact of using beam radiation experiments ensure that realistic GPU FIT rates and error models are reported.

2.2.1 Functional Safety for Automotive Systems

One of the most important features in self-driving cars is the human and obstacles detection. From 2016, only vehicles implementing pedestrian detection are eligible to receive five stars (which is the highest possible score) in the security evaluation from the European New Car Assessment Program (Euro NCAP), which is the most reliable car safety evaluation agency in Europe (PROGRAMME, 2012). In this scenario, it is fundamental to test detection algorithms and validate their reliability.

To be applied in automotive applications, any system must be compliant with the strict ISO26262 constraints (DONGARRA; MEUER; STROHMAIER, 2015). For safety-critical automotive applications, like the autonomous driving or the pedestrian detection, the requirements are identified with Automotive Safety Integrity (ASIL) level D, which is the highest classification of injury risk. ASIL level D is the most stringent level of safety measures to be applied to avoid an unreasonable residual risk. It imposes any component of the system to be able to detect 99% of permanent and transient faults. The allowed hardware failure probability (either radiation-related or not) is limited to 10 Failures In Time (FIT, i.e., errors in 10^9 hours of operations).

Such a requirement is extremely restricted. A simple comparison can be made to have a reference, considering the fatal crashes as the FIT rate that occurred in the United States in 2013, which The National Highway Traffic Safety Administration (NHTSA) estimates to have been 30,057. Of those, 10,076 were driver related fatal crashes which may have been avoided with a reliable self-driving or crash warning system. The non-fatal crashes were 5,657,000 (DONGARRA; MEUER; STROHMAIER, 2013). To measure the FIT rate, besides the number of events, it is necessary to calculate the number of hours of operations (i.e., the total number of hours driven in 2013). For NHTSA, the number of Vehicle Miles Traveled in 2013 was about 3×10^{12}. Assuming conservatively that those miles were traveled with an average speed that ranges between 25 and 50 Miles Per Hour (MPH), the accidents occurred in a time window between 12×10^{10} and 6×10^{10} hours. Considering only the fatal crashes caused by the direct responsibility of the driver as failures that occurred in the measured time window, *fatal crash FIT rate* is between 84 and 168. For non-fatal crashes, the FIT rate would result in about 28,582. As said, the ASIL level D target is 10 FIT. Having self-driving cars fully compliant with ISO26262 would reduce the current fatal crash rate from 9 to 17 times and the non-fatal crash rate more than 3,000 times (SAXENA, 2016). This evaluation on one side demonstrates the importance of reliable autonomous cars and, on the other, highlights the challenging effort required to be compliant with ISO 26262. Unfortunately, today's self-driving cars are still far from reaching the 10 FIT goal. Google reports its first automobile accident caused by system error in February, 2016 (GOOGLE, 2016b). Considering that the Google cars have driven about 1.5×10^{6} miles until June 2016 (as reported in (GOOGLE, 2016a)) their FIT rate, under the assumption of an average speed between 25 and 50 MPH, is between 16,000 and 33,000 FIT, well above the ASIL-D requirements.

3 PEDESTRIAN DETECTION ALGORITHMS RELIABILITY

This chapter serves as a background to understand Pedestrian Detection importance and the algorithms key features. The algorithm analysis is also necessary to comprehend the causes of observed errors and choose the critical procedures to be hardened. Then, the related works in the field of pedestrian detection quality are reviewed, and functional safety for automotive systems is discussed.

3.1 Reliability of Object Detection Algorithms

Embedded GPUs are part of projects implementing the Advanced Driver Assistance Systems (ADAS), which analyzes camera or radar signals to detect pedestrians or obstacles and activate the car brakes to prevent collisions (PROGRAMME, 2012). Additionally, autonomous driving systems, which is the new trend in the automotive market, rely on humans or obstacles computer-aided detection. The reliability characterization of algorithms that implement objects detection is then mandatory to ensure automotive systems safety.

This work considers two classes of Object Detection Algorithm: a Feature Descriptor and Convolutional Neural Networks (CNNs). Feature descriptors and CNN share some similar characteristics, such as both extract information from the input frame by a feature extraction process. They also have on the end of classification task a previously trained classifier to predict the final output. However, the main difference between HOG (Histogram of Oriented Gradients) and CNNs is that HOG extracts a previously defined representation from the image, and CNNs learns the representations by extracting features from the frames.

HOG algorithm (DALAL; TRIGGS, 2005) is the core of several pedestrian or object detection systems (SIVARAMAN; TRIVEDI, 2013; ZHU et al., 2006; KANG; LIM, 2014; LI; GUO, 2013). HOG can be combined with a variety of classifiers to detect pedestrians. In this work, HOG is used as a feature descriptor together with a Support Vector Machine (SVM) as a classifier (DALAL; TRIGGS, 2005). If otherwise stated, in the rest of this work the conjunction of HOG and SVM will just be addressed as HOG.

Two CNNs are considered, Darknet and Faster RCNN. Darknet is an open source convolutional neural network for object classification and detection, written in C and CUDA (REDMON et al., 2015). Faster RCNN is an object detection framework based on convolutional networks written in C++ and Python. The core of Faster RCNN uses Caffe (JIA et al., 2014) a Deep Learning Framework developed by Berkeley Vision and Learning Center (BVLC).

3.1.1 Histogram of Oriented Gradients

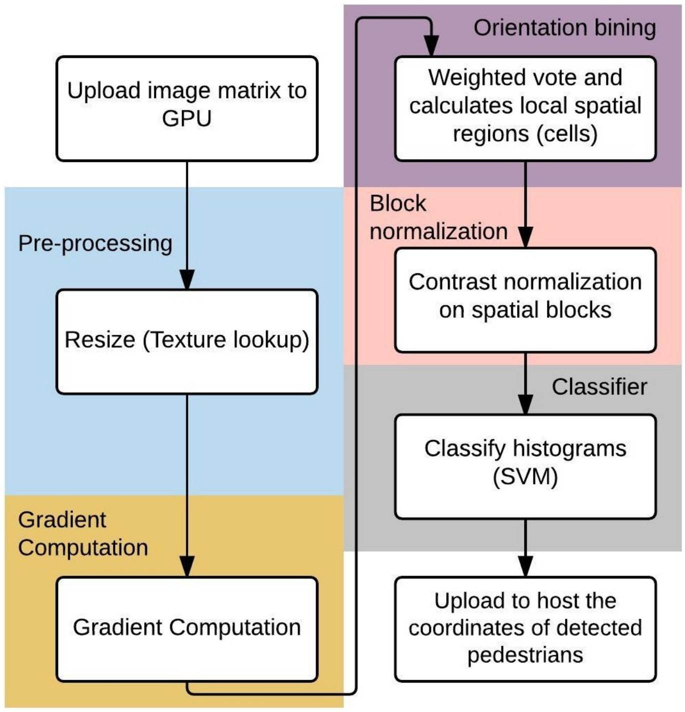
HOG is one of the most common features descriptors for pattern or object detection (YANG; ZHANG; TIAN, 2012; LIM; ZITNICK; DOLLAR, 2013; REN; RAMANAN, 2013), particularly in automotive applications (SIVARAMAN; TRIVEDI, 2013), and its applications on both human (ZHU et al., 2006; KANG; LIM, 2014) and vehicles (LI; GUO, 2013) detection are widespread on today's self-driving vehicles. One of the reasons for HOG popularity is normalization, which eliminates illumination variance on daylight scenes, improving shape detection. HOG orientation sampling is extremely versatile and can be defined to detect different objects. By simply changing the classifier characteristic, HOG can be used to detect humans or objects as well as to recognize faces or gestures.

HOG is used as a feature descriptor, and SVM is used as a classifier. Two main reasons dictated this choice: (1) HOG combined with SVM is one of the most popular and efficient detection systems (DALAL; TRIGGS, 2005; DOLLAR et al., 2012). (2) The availability of HOG together with SVM on a publicly accessible library eases the reproduction of our results (BRADSKI, 2000). While radiation experiments results are strictly related to the chosen feature and classifier, the reliability discussion and fault-injection results for HOG are directly applicable to other classifiers.

On HOG each frame is analyzed by creating a set of blocks, which are pedestrian candidates. Within these blocks, the features are extracted and classified using an SVM, yielding a set of validated BBs (Bouding Boxes). Figure 3.1 shows the OpenCV HOG behavior. The steps showed in the picture are separately analyzed in the following. The fault injection routine described in Chapter 4 is used to identify which step is more critical.

On figure 3.1 the CPU triggers the execution loading to the GPU the image matrix and the data of a previously trained SVM. The method is composed of five GPU steps: Gradient computation, Orientation binning, Descriptor Block and Block normalization (referred only as Block normalization), and Classifier (DALAL; TRIGGS, 2005). The OpenCV version of HOG has a special feature over the original method: an image pre-processing for color correction (see Figure 3.1). Before performing HOG main steps, the

Figure 3.1: OpenCV HOG algorithm execution flow. Adapted from (DALAL; TRIGGS, 2005)



GPU executes a lookup texture on the processing frame.

Gradient Computation is the first step, in which a simple derivative mask filter is applied to the image to detect gradients. The kernel is a very simple mask applied to every pixel on the picture and, as such, intrinsically robust against soft errors. In fact, an error in a step of a filter is averaged with correct values and typically does not significantly affect the output.

The second phase of HOG is **Orientation binning**. In this step, the image is divided into a fixed number of 8x8 pixels regions called *cells*. A histogram of gradient orientations is computed on the pixels within each cell. The histogram is calculated with a weighted vote on the cell, which is a function of the gradient magnitude at the pixel (DALAL; TRIGGS, 2005).

The **Descriptor Blocks and Block normalization** phase groups adjacent cells as spatial regions, called *blocks*, of various dimensions. Cells are grouped based on their gradient orientation, and a cell can participate to more than one block. Once cells are arranged, their histograms are normalized to avoid illumination and contrast bias. A cell can have different normalizations, one for each block it belongs to. Each block is represented by a block descriptor, which is a vector that considers the contributions of all the normalized cells in the block. The final *object* (i.e., a potential BB) detection is based on various blocks descriptors. Blocks typically overlap so that each block may contribute to various *objects*. The next step of HOG classifies *objects* into BBs.

The peculiarity of cells to participate to various blocks and of the *object* to be formed of different blocks could significantly impact HOG behavior under radiation. For HOG it is more likely that a radiation-corruption affects HOG in a way that increases, rather than decreases, the number of BBs, due to the algorithms characteristics. Assuming that a block (or the descriptor of a block) that participates in a BB got corrupted in a way that prevents HOG to consider it as part of the *object*. As blocks overlap, HOG could still identify the *object* without the corrupted block thanks to other correctly computed blocks.

For HOG to miss a BB, radiation should corrupt most of the blocks that compose that BB, which is unlikely. On the other hand, there are two situations in which radiation can induce additional BBs. (1) The corruption of a block that is part of a BB leads HOG to create an additional BB over the same portion of the image or to split the BB into two BBs. (2) The corruption of a block that should not be part of a BB leads HOG to create an additional BB over a portion of the image that does not represent a human (or object). This latter situation may lead HOG to create a BB with the dimension of a single block or to erroneously group the faulty block with adjacent ones to create a larger BB.

The last step of the HOG method is **Classifier**. The soft linear SVM classifier is trained with SVMLight, that solves several problems on SVM vectors classifiers (JOACHIMS, 1999). Humans (or other objects) are identified by BBs, described by four coordinates. A different training could be used to identify other objects in the image. This step is particularly critical for the overall system reliability. Errors in the matrix that results from the SVM training significantly impact the pedestrian detection.

3.1.2 Convolutional neural networks

Artificial Neural Networks (ANNs) are becoming a widely adopted computational approach in many fields, from data mining to pattern recognition, High-Performance Computing (HPC), and data analysis (ESSEN et al., 2015; AMIN; AGARWAL; BEG, 2013). Lately, ANNs have been employed in self-driving cars, which rely on multiple pattern recognition algorithms to identify and classify objects based on captured frames or signals. Most of these algorithms can be efficiently implemented using ANNs (REDMON et al., 2015; JIA et al., 2014; NEAGOE; CIOTEC; BARAR, 2012).

Nowadays Deep Neural Networks (DNNs) are one of the most efficient ways to perform object detection and classification. These networks achieve great performances in visual recognition challenges, like image classification (CIREGAN; MEIER; SCHMIDHUBER, 2012), segmentation (CIRESAN et al., 2012), and object detection (REDMON et al., 2015; SZEGEDY; TOSHEV; ERHAN, 2013), showing significant improvement over other technologies such as WordChannels and SpatialPooling (ANGELOVA et al., 2015). Much research has been done specifically regarding using DNNs to perform pedestrian detection in real time and showed promising results (LUO et al., 2014; RIBEIRO et al., 2016; ANGELOVA et al., 2015).

ANNs have a massively parallel structure and require high computational capabilities. GPUs (Graphics Processing Unit) appear then very suitable to execute ANN algorithms. A particular type of ANNs which can produce significant results for image processing is Convolutional Neural Networks (CNN). Ordinary ANNs are not very scalable for image processing since they are fully connected networks. Considering a picture with 32×32 (32 pixels of width and 32 pixels of height) on RGB (Red, Green and Blue color channels), it would produce an input of $32 \times 32 \times 3$ weights to a fully connected ANN. At a first look this seems manageable but, for bigger images, the size of the network and its weights would add up fast. This fully connected ANN characteristic would be time wasteful and can overfit the ANN weights (when the weights adjust so much to the training dataset that this will interfere on not trained inputs).

To overcome the ANNs limitations, Convolutional Neural Networks have neurons arranged in three dimensions - i.e. width, height and the color channels. CNNs are not entirely connected networks, its starts with convolutional layers and finishes its execution with fully connected layers (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Such CNNs characteristics improve their accuracy and performance on object detection tasks, once it does not produce an amount of unnecessary data and can extract only the important features of the input image. The first CNN proposed was LeNet (LECUN et al., 1998), since then lots of new architectures were created, but they all keep the central concepts of LeNet. Generally, CNNs architectures are composed of the following layers:

1. **Convolutional Layer** performs a convolution on its input. In image processing, a convolution process consists in convolving a filter into an image/matrix - i.e. multiply and sum every filter matrix position by the each image/matrix pixel and store it into a resulting matrix. The resulting matrix of a convolutional layer is called Feature Map since each filter applied to the picture extracts a feature from the input. On CNNs the convolution process could obtain lots of features depending on the number of used filters. The first CNN convolution layer data input are the image

matrices, while the following convolutional layers are fed with the extracted features from the previous ones. As already said, CNNs are not entirely connected, so the output of a convolutional layer will not be completely assigned to the next convolutional layer, that is, some feature maps will be treated differently by the next layer.

2. **ReLU Layer** stands for Rectified Linear Unit Layer which applies a function $f(x) = \max(0, x)$ to every image pixel. ReLU layers remove all non-positive values on feature map gotten from the previous convolutional layer. This type of layer removes the non-linearity of the network, once the real world data is non-linear.
3. **Pooling Layer** Pooling Layer is a layer placed periodically in a CNN to reduce the spatial size of the network. Even if CNNs are not entirely connected, its parameters grow fast in size, so it is necessary to place pooling layers aiming to reduce the size of the features maps. Pooling is made through a function, for example, Maxpooling. Maxpooling splits the matrix into n block regions and then selects the bigger value of each block and places it to the output matrix. The output matrix size will be n times smaller.
4. **Fully connected Layer** is the last step of a CNN, which is in charge of making a classification based on the extracted features on the previous layers. Fully connected layer are regular artificial neural networks, they have full connection to the last layer.

Some CNNs also have normalization layers, which are responsible for normalizing the neurons activation values, applying a transformation that maintains the standard deviation close to 1 and means activation values nearly 0.

Due to CNN characteristics, it is possible to compute its layers as matrix multiplication operation, by a Im2col operation, which transforms kernel convolution operation into a matrix multiplication. Since GPUs GEMM (General Matrix Multiplication) functions are well known for their performance, it is possible to take advantage on object detection on GPUs. However, according to Oliveira *et al.* (OLIVEIRA et al., 2014a), GPU GEMM is particularly prone to experience radiation-induced errors, so it is necessary to measure the reliability strengths and weaknesses of CNNs on GPUs. In the next sections, Darknet and Faster RCNN are described, those networks were tested in radiation experiments. The results for tested CNNs are described on Chapter 5.

3.1.2.1 You Only Look Once framework

$Darknet$ is an open source Convolutional Neural Network (CNN) used to implement the YOLO object detection system (REDMON et al., 2015). Darknet is a Deep Convolutional Neural Network that YOLO uses to perform object detection and classification, it is capable of detecting objects in real time, analyzing up to 20 frames per second (45 frames per second on a smaller version of Darknet). YOLO became famous because of its performances in the PASCAL VOC 2007 Challenge, as it was the only real time system and had a high accuracy compared to the other systems (EVERINGHAM et al., 2010).

Darknet, as most CNNs, performs three operations to object detection, which are: (1) Convolutional layers. (2) Max pooling layers. (3) Classification through a fully connected layer.

YOLO performs its detection by dividing the input image into an $S \times S$ grid and then calculating the class probability for each grid cell, as well as the bounding boxes and confidence. Hence, it can detect and classify objects in real time. The network uses 24 convolutional layers, which are then followed by two fully connected layers. There are also four maxpool layers in specific stages of the network.

Figure 3.2 shows the architecture of Darknet, it has an unusual characteristic compared to other detection systems as it has a single execution pipeline. While other (UIJLINGS et al., 2013; LOWE, 1999; PAPAGEORGIOU; OREN; POGGIO, 1998; SERMANET et al., 2013) detection systems first generate a list of probable objects, a CNN extracts the features, a nonlinear model adjusts the bounding boxes, and a nonmax suppression algorithm removes the duplicated bounding boxes, YOLO performs all of it in a single execution (REDMON et al., 2015). A single execution pipeline could be good for the system reliability since it will stress less the caches and the functional units.

However, the lack of complex pipeline stages could prevent errors in the first stages to be masked by following iterations.

3.1.2.2 Python Faster RCNN

Faster RCNN is an improvement of older versions of RCNN (REN et al., 2015). The authors changed the complex RCNN pipeline to add more performance and precision to Faster RCNN. Figure 3.3 shows Faster RCNN's execution pipeline, the network has two branches to add more detection precision. On the first stage, a traditional convolu-

Figure 3.2: Darknet network architecture. Adapted from (REDMON et al., 2015)

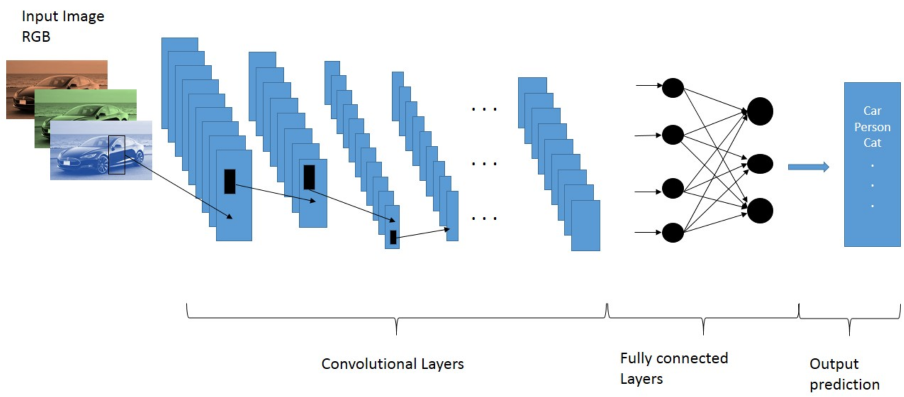
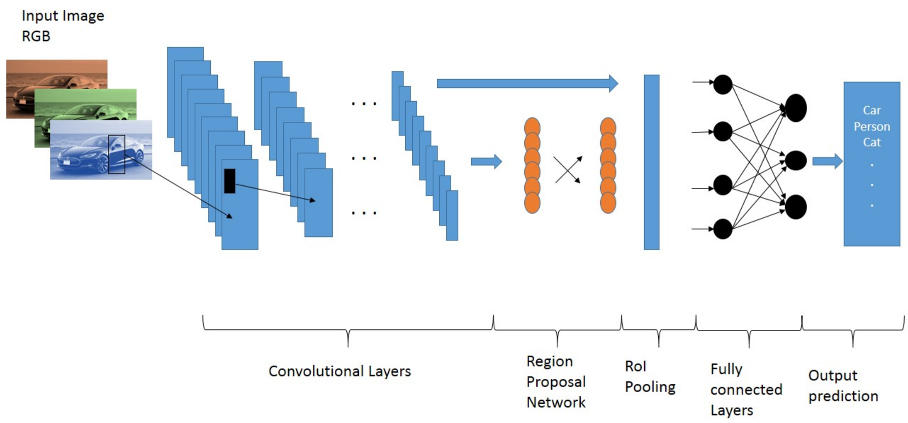


Figure 3.3: Faster RCNN network architecture. Adapted from (REN et al., 2015)



tional network proposes regions on the input image and produces feature maps.

On the first branch, after the convolutional layers, a Region Proposal Network (RPN) slides a window over an input feature map, and then it will "anchor" a set of scores (i.e. are the bound boxes probabilities of that region) and a set of bounding boxes coordinates for the processing window. An RPN is a network which takes an image/feature map as input and outputs the rectangular object proposal i.e. possible objects on the image. The set of proposed regions will then go through a RoI (Region Of Interest) pooling layer, which is a maxpooling operation for only the proposed regions. Finally, RoI output will be evaluated by a fully connected layer.

The second Faster RCNN branch gets the convolutional layers output and places it as an input to the RoI pooling layer. Its output will also be evaluated on the fully connected layer, so the bounding boxes and scores will be a combination of the classification made on the fully connected layer, using the two previous steps outputs.

Faster RCNN architecture seems to be very reliable since it processes the RPN output on the fully connected layer using a region of interests. Even if an error occurs on one of the cited stages (RPN or RoI pooling), it may be overwritten by the classification stage, once it computes the predicted bounding boxes and theirs scores based on precomputed regions of interest, which is composed by various anchor boxes and anchor scores. Even an error in a region of interest that modifies the anchor boxes, and/or the way an anchor box contributes to the object, it could result in a non-critical error.

Faster RCNN is built using *Caffe* and *cuDNN*. Caffe (JIA et al., 2014) is a Deep Learning Framework which has support to be built with NVIDIA cuDNN library. cuDNN is a GPU-accelerated library of primitives for deep neural networks (CHETLUR et al., 2014). Compared with common GEMM frameworks (i.e., cuBLAS) cuDNN has specific optimizations for Neural Networks, which lead to less memory overhead caused for traditional CNNs operations (CHETLUR et al., 2014).

4 FAILURE IN TIME AND ERROR CRITICALITY EVALUATION

This chapter presents the object detection algorithm reliability and the metrics necessary to analyze the object detection algorithms corrupted output criticality. Traditionally, the radiation-induced error rate is calculated considering any SDC or Crashes as failures. Any result produced by the tested device which deviates from the expected one is marked as erroneous and considered in failure rate calculation. The intention of this work is to go beyond the traditional bit-per-bit comparison between the experimental output and the radiation-free one. Two metrics that derive from image processing community were selected to qualify the object detection algorithm radiation-induced corruptions: Precision and Recall. The importance of each metric on the reliability evaluation of the object detection algorithm depends on the application or on the system in which they are applied. As described on Chapter 5, some metrics are of particular importance to identify critical errors and give crucial insights to the device or application designers to improve the reliability of their product.

4.1 Failure in Time

Cross Section is the most common metric to evaluate a system reliability, expressed with the unit of area: cm^2. Cross Section is the circuit area that, if hit by an impinging particle, will probably generate a failure. The higher the Cross Section value the higher the probability for a particle to generate an error.

To evaluate the application sensitivity it is necessary to calculate the cross section dividing the number of observed errors by the received neutron fluence ($neutrons/cm^2$). As such, cross section comes with the unit of an area (cm^2). By multiplying the cross section with the expected neutron flux at which the device will operate, it is possible to estimate the realistic error rate, expressed in Failure In Time (FIT), i.e. errors per 10^9 hours of operation.

4.2 Experimental Methodology

In this work accelerated radiation testing is used to evaluate object detection reliability on embedded GPUs. Then, extensive fault injection campaigns are performed to have insights about the code sensitivity and detect critical portions of the algorithm.

Both radiation experiments and fault-injection information are required for the scope of this work. The gathering of realistic data is essential to precisely evaluate the radiation reliability of the object detection algorithms executed on embedded GPUs and analyze the criticality of radiation-induced errors. Such evaluation, for those devices like COTS for which a Register Transfer Level (RTL) description is not available, can be performed only through radiation experiments. In fact, without an RTL description, faults can be injected only on a selection of user-accessible resources, while radiation testing does not restrain faults to a single part of the chip. Additionally, fault-injection results could be biased on the selected fault-model while radiation experiments mimic the realistic probabilities and manifestations of failures, as described in the Radiation Hardness Assurance (RHA) testing procedure (HERRERA-ALZU; LOPEZ-VALLEJO, 2014).

Radiation experiments provide a realistic mimic of the environment and offer a plethora of advantages. However, it lacks error propagation visibility. It is very hard to correlate SDCs or Crashes with a set of resources or algorithm procedures whose corruption leads to the observed failure. On the contrary, fault-injection can correlate SDCs or Crashes with the root causes in some part of the algorithm. Extensive fault injection campaigns are performed to understand errors propagation in this work better. In fact, it is necessary to identify the object detection algorithm's critical procedures to deeply analyze its criticality and provide insights for designing efficient hardening solutions. The intrinsic limitations of fault-injection on COTS make it impossible to correlate each physical radiation-induced error with its manifestation at the output. However, by corrupting variable, registers, and instruction output values, it is possible to identify those parts of the code that are likely to affect the object detection algorithm execution (HARI et al., 2014).

The proposed fault injection cannot calculate the SDC rate for the object detection algorithms. However, it can be used to evaluate the percentage of injected errors that caused SDCs. By injecting fault at hardware or software level, it is possible to measure the AVF (Architectural Vulnerability Factor), which is the probability for a low-level corruption to propagate until the output (MUKHERJEE et al., 2003). At the software level, it is possible to calculate the PVF (Program Vulnerability Factor), which is the probability of a fault at the instruction level to affect the program output (SRIDHARAN; KAELI, 2009). The correlation of fault-injection and radiation experiment outputs allows to identify which procedures or variables are more likely to generate the observed critical failures and, thus, should be hardened.

4.3 Devices Under Test

Table 4.1 shows all tested GPUs, aiming to obtain significant statistical data, five different GPU architectures are tested. Three commercially available embedded (*Tegra K1*, *APU A10-7850K*, and *Tegra X1*) GPUs architectures and two HPC ones (*K40* and *Titan X*). Two main reasons dictate the choice of the HPC GPUs architectures. (1) *K40* is the latest NVIDIA GPU made with 28nm CMOS technology. It is worth to make a comparison between *Tegra X1* and *K40* since *Tegra X1* is the most advanced embedded GPU until this work is done. (2) *Titan X* is built with the novel Pascal architecture, which has a particular architecture for Deep Neural Networks. Another reason for *Titan X* on this work is that the next generation of embedded GPUs will be developed with Pascal architecture, as well as *Titan X* (ANANDTECH, 2016).

APU A10-7850K: this architecture includes 4 CPU cores, designed with an operating frequency of up to 4.1 GHz. The CPU has 256 KB of L1 data and instruction cache and 4MB of L2 cache. The general processor power consumption is about 95W. The *APU A10-7850K*'s embedded GPU has 512 cores, divided into 8 Compute Units with an operating frequency of about 866MHz. *APU A10-7850K*'s main memory is DDR3 with a frequency of 2.133 MHz, using at most two channels. The GPU can execute up to 512 parallel threads, and the CPU up to 4 threads.

Tegra K1: this embedded SOC is powered by an ARM quad-core Cortex-A15 from ARMv7 family, with an operating frequency of 2.3 GHz. The ARM core disposes of 32KB of L1 data and instruction cache, and 2MB shared L2 cache. The embedded GPU has 192 computing cores with a 64KB of a register file. As main memory *Tegra K1* disposes of 2 GB of DD3L, operating at 933 MHz. *Tegra K1* SoC is available on Jetson K1 development kit, which is an NVIDIA platform for embedded system developers (NVIDIA, 2014).

K40: GPU is designed with $Kepler$ architecture in a 28nm standard CMOS technology. *K40* has 2880 CUDA cores, divided into 15 Streaming Multiprocessors (SMs), with 192 CUDA cores for each SM. The register file and L1 cache are shared between SM's cores, on *K40* each SM has 64K registers, 64KB of L1 cache memory, and 48KB of read-only data cache. SMs share 1.5MB of L2 cache and a total 6GB GDDR5 memory. The register files, shared memory, L1 and L2 caches are SECDED protected, while read-only data cache is parity protected.

Tegra X1: is an embedded SOC designed with a $Maxwell$ based GPU and an ARM quad core. It is built in a 20nm standard CMOS technology. The *Tegra X1* GPU has 256 CUDA cores, which are divided in two SMs. Each Maxwell core can deliver up to 40% greater performance when compared to a Kepler CUDA core, and up to twice the performance per watt. Each *Tegra X1* GPU SM has 64KB of L1 cache and 32KB of registers file capacity, and 256KB of L2 cache shared between SM's cores. The *Tegra X1* operates at 1GHz, achieving performances of up to 1 Teraflop using FP16 (half floating point precision) operations (NVIDIA, 2015). On the CPU side, a quad-core ARM A57 is the general purpose processor. Each A57 core has 48KB of L1 instruction cache and 32KB of the L1 data cache. The L2 cache has 2MB shared between the four A57 cores (NVIDIA, 2015). *Tegra X1* SoC is available on Jetson X1 development kit, which is an NVIDIA platform for embedded system programers (NVIDIA, 2015a).

Titan X: is designed with the novel $Pascal$ architecture, with a 16nm FinFET technology. It has 3584 CUDA cores split into 28 SMs. Titan X was explicitly designed to be more efficient in executing neural networks operations. NVIDIA *Titan X* has much more memory compared to the others GPUs, it has 12GB of GDDR5X SDRAM memory. Each SM shares a register file with 256KB and 48KB L1 cache. L2 cache has 3MB capacity divided between all SMs. *Titan X* power supply is 250W, with 1.4GHz of GPU base clock (NVIDIA, 2016).

Despite *K40*, none of the mentioned GPUs offer any reliability solution like ECC. Testing an SECDED protection is essential to evaluate how this reliability technique can prevent SDCs and the implications of this technology on system crash.

APU A10-7850K, *Tegra K1* and *Tegra X1* are designed with an SIMD (Single Instruction Multiple Data) architecture. SIMD processor allows a single instruction to operate on a group of data in parallel (HENNESSY; PATTERSON, 2011), commonly arrays and matrices. A significant number of processing units is required to permit programmers to extract parallelism from SIMD architecture and speed up execution. Since modern embedded GPUs can provide hundreds of processing units, it is easy to achieve high speedups on Computer Vision applications that require an extensive use of operations on arrays and matrices, like Object Detection.

Table 4.1: Comparison between all tested devices

	CPU Architecture	GPU Architecture	CUDA Cores	Transistor Size	Cache	SMs	Power Consumption
Tegra K1	4 ARM Cortex-A15 32 bits	Kepler GK20A	192	28 nm	L1 instruction/data cache 64KB; L2 Unified Cache 2MB	1	up to 5W
APU A10-7850K	AMD Kaveri	Radeon 7	512	28 nm	L1 total cache 256KB; L2 total cache 4 MB	8	95W
K40	–	Kepler GK110B	2880	28 nm	L1 cache 64KB; L2 cache 1.5MB	15	235 W
Tegra X1	ARM Cortex-A57 64 bits	Maxwell GM20B	256	20 nm	L1 instruction cache 48KB and data cache 32KB; L2 Unified Cache 2MB	2	up to 10W
Titan X	–	Pascal GP102-400	3584	16nm	L1 total cache 48 KB; L2 total cache 3MB	28	250W

4.4 Experimental Setup

Radiation experiments were performed at the ChipIR facility at the Rutherford Appleton Laboratory (RAL) in Didcot, UK and the Los Alamos National Laboratory (LANL) Los Alamos Neutron Science Center (LANSCE) Irradiation of Chips and Electronics House (figure 4.2). ChipIR and LANSCE provide a white neutron source that emulates the energy spectrum of the atmospheric neutron flux between 10 and 750 MeV.

The available neutron flux was between 10^6 and $10^7 n/(cm^2 \times s)$ for energies above 10 MeV. The neutron flux the GPUs receive during experiments is ten orders of magnitude higher than the atmospheric neutron flux (which is $13n/(cm^2 \times h)$ at sea level (JEDEC, 2006)). The experiments were carefully designed to ensure that the probability of more than one neutron generating a failure in a single code execution remains practically negligible. The observed error rates were lower than 10^{-3} errors/execution. Since a much lower neutron flux may hit a GPU in a realistic environment, it is highly likely not to have more than one corruption during a single execution. Therefore, it is possible to scale the experimental data in the natural radioactive environment without introducing artificial behaviors (VIOLANTE et al., 2007).

The experimentally observed cross-section gains importance as it becomes an intrinsic characteristic of the device and application, independent of the neutron source. Multiplying the experimentally measured cross-section (cm^2) by the expected neutron flux on the SoC ($13n/(cm^2 \times h)$ at sea level (JEDEC, 2006)), one can estimate the realistic FIT rate of the device executing the application.

For all experiments, the beam was restricted to a spot with a diameter of 2 inches, which is enough to irradiate *APU A10-7850K*, *Tegra K1*, *Tegra X1*, *K40* adequately, and *Titan X* chips without directly affecting nearby board power control circuitry and DRAM chips. To ensure that data stored in the main memory is not corrupted, allowing an analysis focus on the devices' core reliability, a software watchdog was set to checks errors in the DRAM.

Figure 4.1: LANSCE and ChipIR (ISIS) neutron spectrum compared to the terrestrial one.

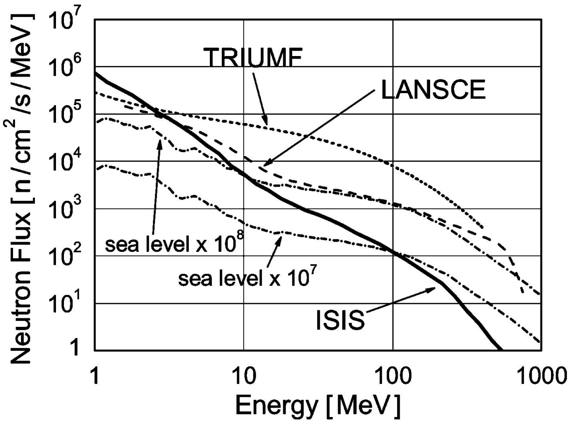
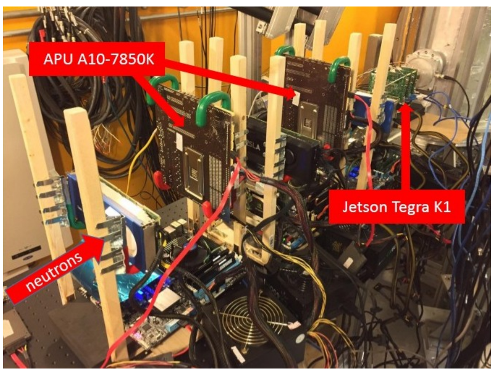


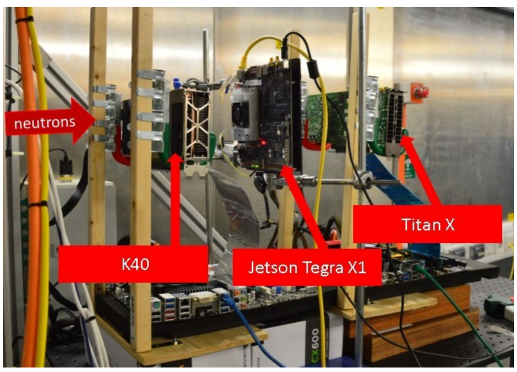
Figure 4.2a shows HOG setup for two *APU A10-7850K* and four *Tegra K1* GPUs, aligned with the beam to reduce statistical error. The chips were then placed at different distances from the neutron source. A de-rating factor was applied to consider distance attenuation. After the de-rating, the devices error rate seemed independent on the position, suggesting that the neutron attenuation caused by other boards between the source and the device under test is negligible.

Figure 4.2b shows part of the experimental setup mounted at LANSCE for CNNs on *K40*, *Tegra X1* and *Titan X*. A host computer initializes the test sending pre-selected input to the GPU and gathers results, comparing them with a pre-computed golden output. It is worth noting that this comparison takes no more than 10% of the total execution time. When a mismatch is detected, the execution is marked as affected by a *Silent Data Corruption (SDC)*.

Figure 4.2: (a) The experimental setup at LANSCE. A total of 2 *APU A10-7850K* and 4 *Tegra K1* embedded GPUs were aligned with the neutron beam. The beam direction is indicated by the arrow. (b) Radiation test setup inside the ChipIR facility, RAL, Didcot, UK.



(a)



(b)

Software and hardware watchdogs were included in the setup. The software watchdog monitors a time-stamp written by the application under test. If the timestamp is not updated in a few seconds, the kernel is killed and launched again. This watchdog detects *Crashes*, i.e. application crashes or control flow errors that prevent the GPU from completing assigned tasks (e.g., an infinite loop). The hardware watchdog is an Ethernet-controlled switch that performs a power cycle of the host computer if the host computer itself does not acknowledge any ping requests in ten minutes. The hardware watchdog is necessary to detect *Hangs*.

Each application under test (i.e., the code that is continuously executed by the processor after the initialization phase) has three different stages, which form an *application execution* (see Algorithm 1). First, input data are initialized with pre-determined values; then, the application realizes its functional operation; Lastly, the result of the operation is compared to a golden copy, and success or failure is reported. After the last phase, a new execution is triggered. It is worth noting that the input data initialization and the comparison with the golden copy represent only a small part of the execution time ($< 10\%$).

```
while True do
    // Application Execution Start
    input = data_initialization();
    result = functional_operation(input);
    if result == golden_copy then
        print(Ok);
    else
        print(SDC_detected);
        download(corrupted_output);
    end
    // Application Execution End
end
```

Algorithm 1: Application under test. After initialization, application executions are continuously triggered.

4.5 Metrics for error evaluation

The analysis goes a step beyond the traditional SDCs detection by considering error criticality. Besides measuring the number of radiation-induced output errors, it is also necessary to qualify the effect of observed SDCs on the pedestrian detection reliability. This work considers Precision and Recall. Correlating the selected metrics, it is possible to distinguish those critical errors that could prevent a pedestrian detection or erroneously lead the vehicle to a sudden stop.

4.5.1 Precision and Recall

In the image processing community Precision and Recall are widely used to compare the BBs identified by the algorithm with the ground truth, so to assess the quality of a given classifier (FAWCETT, 2006). Employing Precision and Recall to compare the experimental output with the radiation-free output (not with the ground truth) is necessary to evaluate how radiation affects the object detection output.

Precision and Recall are given by:

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

where TP is the number of *True Positives* (objects that were detected), FP is the number of *False Positives* (outcomes of the classifier that do not correspond to an object), and FN is the number of *False Negatives* (an object that was not detected by the classifier). Experimental results are qualified considering a BB n in the radiation-corrupted output as TP if, for any BB m of the radiation-free output the following condition is verified:

$$Jaccard\_similarity(n, m) > T_J \tag{4.3}$$

$where$:

$$Jaccard\_similarity(n, m) = \frac{|m \bigcap n|}{|m \bigcup n|} = \frac{|m \bigcap n|}{|m| + |n| - |m \bigcap n|} \tag{4.4}$$

where T_J is the acceptance threshold. Otherwise, n is considered as an FP. If for a given BB m of the radiation-free output there is no BB n on the corrupted output which satisfies this condition, an FN is detected.

T_J is an arbitrary threshold, with $0 \leq T_J \leq 1$. Values of T_J close to 1 impose the classifier to be extremely precise, i.e. BBs detected by the algorithm have to match the ground truth exactly. Values of T_J closer to 0 relax detection precision. $T_J = 0.5$ has been identified as a good trade-off to evaluate detection quality, as it allows some detection imperfection yet maintaining a good relation with the ground truth (FAWCETT, 2006). When evaluating the impact of radiation on object detection algorithm, T_J becomes a threshold to distinguish between critical and noncritical corruptions. Any corruption that preserves Equation 4.3 is not considered critical, as BBs are to be regarded as sufficiently similar to the radiation-free output. When Equation 4.3 is not verified, it means that radiation induces an FP or FN (i.e., additional or missed objects, respectively). As such, $T_J = 1$ imposes any radiation-induced error to be marked as critical while values of T_J close to 0 allow most corruptions to be acceptable. In the reliability context $T_J = 0.5$ still holds, as the T_J trade-off discussion presented in (FAWCETT, 2006) is valid independently from the source of detection imprecision (intrinsic algorithm detection imprecision or radiation-induced corruption). Then, to compare the object detection algorithm radiation corrupted output with the radiation-free output the $T_J = 0.5$ is selected.

The Recall rate provides the fraction of existing objects that were detected by the classifier, even in the event of a radiation-induced error. Hence, $R = 100\%$ means that all objects were successfully detected. On the other hand, Precision measures the fraction of the detections produced by the classifier that relate to an object, so that $P = 100\%$ means that all detections generated by the classifier correspond to objects. In general, the detection imprecision (either radiation related or not) empathizes Precision where expenses Recall or vice-versa. An error could then lower Recall (i.e., miss some objects) but increase Precision (i.e., reduce the number of false positives) or the other way around.

4.6 Applications Under Test

The quality of classifiers is proved using videos from available datasets. Such proof is required as classifiers are not perfect, and their performances depend on the selected frames. Researchers are then aiming at improving the average performances of their algorithms on several representative frames. The purpose of this work is a little bit different. This work characterizes the radiation impact on two different object detection algorithms, i.e., if their proprieties are affected by radiation on a radiation-free execution (not on the ground truth). It is worth noting that the radiation-free execution may differ from the ground truth. However, the evaluation of object detection algorithm quality is out of the scope of this work as it has already been deeply investigated (DALAL; TRIGGS, 2005; REDMON et al., 2015; REN et al., 2015). Table 4.2 shows Frames Per Second (FPS) and detection precision for each tested platform and benchmark. It is remarkable that FPS for *Tegra K1*, *APU A10-7850K* and *Tegra X1* are smaller than *K40* and *Titan X*, due theirs carachteristics.

Table 4.2: Frames Per Second and Detection precision

	General detection precision [Hit rate]	*Tegra K1* [FPS]	*APU A10-7850K* [FPS]	*Tegra X1* [FPS]	*K40* [FPS]	*Titan X* [FPS]
Darknet	$\simeq 66\%$	–	–	3.07	10.43	15.46
Faster R-CNN	$\simeq 73\%$	–	–	0.41	3.65	7.83
HOG	$\simeq 50\%$	0.77	2.48	–	–	–

4.6.1 HOG

The experimental and analytical study on HOG algorithm is done using a version implemented on the OpenCV library, in its 2.4.9 version. The selected version has an open source sample test to allow developers to set the method's parameters. The *number of levels* was set to 100, *hit threshold* to 0.8, and *group threshold* to 1, following OpenCV directives for frames in which some objects can be covered by more than one BB (BRADSKI, 2000).

For the aims of the proposed study, it is sufficient to run HOG on a single frame, as long as the frame is sufficiently complex and representative.

The chosen static frame is illustrated in Figure 5.5a. The image is sufficiently complex to induce HOG to create several BBs of different sizes and positions, and also includes clusters of pedestrians, cars, and other objects. This single frame allows us then to evaluate how radiation impacts HOG detection in a wide set of situations. The chosen frame is specifically selected for radiation experiments and is not part of data sets commonly used to evaluate detection quality. This choice is dictated by the fact that frames on available data sets, while universally used to evaluate detection quality, may be too simple to evaluate HOG reliability.

4.6.2 Darknet and Faster RCNN

Darknet and Faster RCNN networks are chosen because of their importance on object detection field (REDMON et al., 2015; REN et al., 2015). Both methods Darknet and Faster RCNN can achieve 66% and 73% of correctness on object detection (see table 4.2). The selected CNNs are significant as object detection benchmarks until this work were the most advanced CNNs on GPUs.

As already said every CNN iteration the radiation free output and the output produced on radiation environment are compared. Once Darknet and Fast RCNN outputs are float point matrices, an $errorthreshold$ must be set. So when CNNs output matrices are compared it is considered as an SDC if and only if its difference is greater than $errorthreshold$. In this work, the chosen $errorthreshold$ is 5×10^{-3}, the choice of this value is necessary to identify those radiation-induced errors that impact the final object detection.

For CNNs radiation test two notable datasets are chosen. The PASCAL VOC 2012 Classification/Detection Dataset (EVERINGHAM et al., 2010) and the Caltech Pedestrian Dataset (DOLLAR et al., 2009). PASCAL VOC is chosen because it is a renowned dataset and because it was used to evaluate Darknet detection capabilities (REDMON et al., 2015). It consists of more than ten thousand images in which the system should detect and classify objects of 20 predetermined classes, determining their bounding boxes and classes. Caltech is then chosen because of its relevance in today's researchers in the automotive area.

4.7 Fault-Injection

As already mentioned radiation experiments are very reliable methods to measure SDCs and Crash cross section. But to correlate errors to a particular resource or code region it is necessary to employ fault injection. In this work, two types of fault injection are used, one through NVIDIA CUDA debugger and the other with NVIDIA SASSIFI.

4.7.1 GDB Fault Injection

On HOG a fault injection campaign was performed to identify those portions whose corruption is likely to affect pedestrian detection in a critical way. It is worth noting that in cost-sensitive domains, such as the automotive sector, where efficiency regarding per-unit-prices is an essential criterion, full hardware redundancy or dedicated hardware modifications are to be considered too expensive. Moreover, in real-time systems, full-time redundancy overhead may impede to meet deadlines. Our analysis serves as a reference for the design of efficient and effective selective hardening solutions.

Fault-injection campaigns were performed for specific kernels regions of the main HOG phases (detailed in Chapter 3). Using a Python script and CUDA-GDB, it is possible to freeze the HOG execution flow and change local variables values on GPU side routines. The fault injection routine was performed using a methodology similar to GPU-

Qin (FANG et al., 2014b).

All kernel variables are mapped into a list before fault-injection starts. Then a line on one of the five HOG kernels (see Chapter 3) to place a GDB breakpoint, is randomly chosen. When fault-injection starts, the program executes normally until the breakpoint is reached. At the breakpoint execution is frozen and a variable from the local kernel variables list is randomly chosen. The context is switched to the host, which performs error injection by assigning a random value to the selected variable. The breakpoint is then deleted and execution continues. When radiation generates a bit flip (or multiple bit flips) on low-level resources, the bit flip may propagate resulting in a wrong value written to memory. From a high-level view, the wrong variable value is not necessarily limited to a single bit of difference from the correct value. As faults were injected in high-level variables, injecting random values (which includes but is not restricted to single bit flips) is the fault model that better fits the purpose of this work. Thus, a random value generator is used, injecting random values rather than inject only single bit flips.

While the main purpose of the fault injection campaign is to understand the observed radiation-induced errors better, the derived insights could be extended to other sources of SDC. In fact, the observed HOG behaviors when faults are injected in variables does not depend on the physical source of error.

4.7.2 SASSIFI Fault Injection

SASSIFI injects transient errors in Instruction Set Architecture (ISA) visible states such as general purpose registers, stored values, predicate registers, and condition registers (HARI et al., 2015). SASSIFI is divided into three main steps: profiling the kernel application, generating the error injection sites, and injecting fault. SASSIFI is based on SASSI, which is an instrumentation tool that operates at the final step of NVIDIA compilation flow (STEPHENSON MARK SASTRY HARI et al., 2015). SASSIFI then does not disrupt the perceived final instructions schedule or register usage. SASSIFI instructs SASSI on which instrumentation to use and on where to insert it. For profiling/fault-injection, SASSI must instrument all instructions that modify registers or memory. After an instrumentation, SASSI calls a user-defined function which handles the profiling/fault-injection procedure. Since SASSIFI does not need to switch context to the host to inspect or modify GPU state, it introduces a slight time overhead. On the average, SASSIFI fault-injection overhead is less than $5\times$ the regular code execution.

In this work, SASSIFI is used to inject faults into two injection sites, the instruction output (INST) and in the register file (RF). These two injection sites are described as follows:

- **INST** injection site is used to inject faults on the instruction output. INST injections are interesting to measure how transient errors that modify the result of an instruction propagates at the architecture level till the output. SASSIFI takes track of the instruction whose corruption generate the observed error, allowing a detailed study on low-level instructions reliability.
- **RF** injection site is used to inject faults on the register file. With RF it is possible to measure register file AVF and how applications digest an error in memory elements. The difference between INST and RF is used to evaluate ECC effectiveness.

Before injecting faults SASSIFI needs to profile the target application, SASSIFI will profile the GPU kernel instead all program. The profiler steps consist in instrumenting all instructions of the kernel code, so the registers and memory information will be passed to SASSIFI handler to save it in a profiler output file. This phase is mandatory to identify how many injection points exist on the GPU kernel. SASSIFI is used to injected at least 2,000 faults for each error site on HOG and Darknet (only on open source kernels), which are sufficient to guarantee the worst case statistical error bars at 95% confidence level to be at most 1.96%.

CUDA-GDB based fault injections are very slow, depending on the benchmark complexity it could reach a nonpractical time to get a significant statistical data. HOG has a simpler code compared to Darknet, that is, the code size is smaller and has fewer submodules. So a fault injection campaign on Darknet using CUDA-GDB will not be so accurate as HOG, once all complex Darknet code must be mapped. With SASSIFI a real statistical data could be obtained with less time and effort.

5 RESULTS AND DISCUSSION

This chapter reports results obtained through radiation experiments. The experimental results are presented for five different GPU architectures. Then, the metrics discussed in Chapter 4 are applied to radiation induced SDCs to qualify HOG and CNNs corruption, aiming to distinguish between tolerable and critical errors. Fault-injection results are then reported to identify critical procedures. Finally, experimental and analytical analyses are made to propose hardening solutions to increment HOG and CNNs reliability in embedded SoCs.

5.1 HOG and CNNs FIT analysis

To prevent leakage of business sensitive data, the absolute FIT rates can not be shown, so the results presented on figure 5.1 are all normalized using the same constant. However, we can state that SDC FIT rate for all tested platforms is in the order of tens of FITs, above the limit of 10 FIT imposed by ASIL-D. Figure 5.1 shows the results for each execution, the data calculated was compared bit-per-bit with the result of the radiation-free execution. The application execution was then classified as follows:

Correct: The application produced exactly the expected output of a fault-free environment.

SDC: The application produced a different output than a fault-free-environment. This corrupted output is stored and post-processed.

Crash: The system hanged or crashed and had to be restarted, or the system rebooted by itself.

Figure 5.1: SDC and Crash normalized FIT for all tested architectures and configurations.

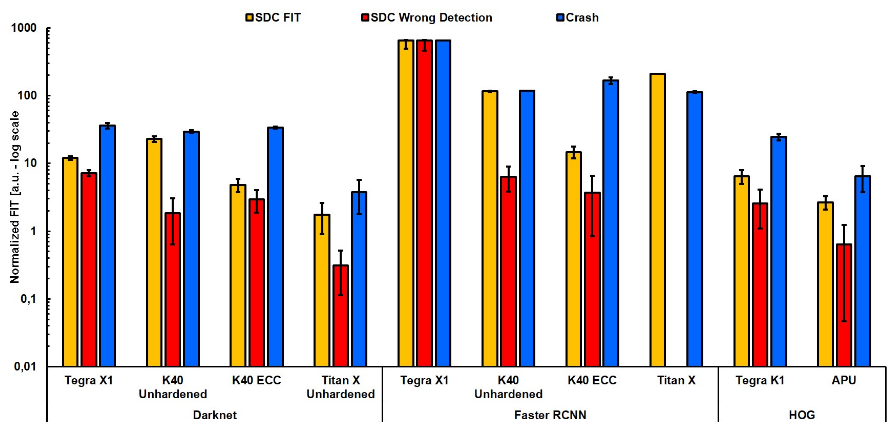


Figure 5.1 also shows the SDC FIT for those executions which produced SDCs that have Precision and Recall values different of 1. All SDCs that have Precision and Recall values different of 1 are considered a critical SDCs, and referred in figure 5.1 as an SDC Wrong Detection. A criticality analysis is better discussed in section 5.2.

5.1.1 HOG

For HOG the Crash rate was of 1.25×10^{-3} *crashes/executions* for *APU A10-7850K* and 6.93×10^{-3} *crashes/executions* for *Tegra K1* SoCs. It is not surprising that for both *APU A10-7850K* and *Tegra K1* Crashes are more likely than SDCs. HOG, in fact, acts as a filter and operates image processing, which is intrinsically robust against SDCs (BREUER; GUPTA; MAK, 2004). As a result, radiation errors that affect data memory elements or operations could be masked during the computation, not affecting the output. In other words, the SDC rate of HOG is likely to be small. The last three steps of HOG (Orientation Binning, Block Normalization, and Classifier) have several control flow operations whose corruption is liable to modify the algorithm flow, eventually generating infinite loops or illegal memory accesses, which lead to Crashes.

It is worth noting that a significant component of Crashes is caused by radiation-corruption of the device control circuitry. Errors affecting instruction memory, the GPU hardware schedulers, or the CPU-GPU interface could lead to the application crash or system hang, independently of the algorithm proprieties (OLIVEIRA et al., 2016). Crashes, while more frequent than SDCs in HOG, are considered less critical as they are easily detected (LI et al., 2008; NAKKA et al., 2005; PATTABIRAMAN et al., 2006). Crash detection and recovery in a real-time system, however, must be quick enough to allow the system to recover without missing deadlines (CANDEA; FOX, 2001; LEE; SHA, 2005; WU; KUO; CHANG, 2006).

HOG is found to be more prone to be corrupted when executed on *APU A10-7850K*. As discussed in Section 5.2.1, *APU A10-7850K* is also more susceptible to critical errors than *Tegra K1*. The most likely reason for the observed discrepancies relies on the number of active parallel processes, which is greater on *APU A10-7850K*. It means that a higher amount of resources will be shared on *APU A10-7850K* concerning *Tegra K1*. Corruption of those shared resources is likely to affect several parallel processes, eventually producing a critical error. Other reasons could be the different transistors layout, many parallel processes scheduling, and different computing units designs. Unfortunately, as both *APU A10-7850K* and *Tegra K1* designs and characteristics are proprietary it is not possible to exhaustively understand the reasons for the observed different behaviors under radiation.

Experiments highlighted a critical *Tegra K1* and *Tegra X1* SoCs configuration. Tegra development boards rely on embedded flash memory to store the system boot-loader. Under radiation, this choice does not seem reliable. In fact, several times during the experiments, the flash memory got corrupted, impeding the correct system boot. To have *Tegra K1* and *Tegra X1* functionality restored it was necessary to re-flash the embedded memory. A boot-loader corruption is detected with two subsequent activations of the hardware watchdog, which takes at least 20 minutes. This impedes a precise measure of the bootloader FIT rate. However, FIT rates for Flash memories of several vendors and technologies are plainly available (JUST et al., 2013; BAGATIN et al., 2014; FOGLE et al., 2004) and are to be used as a reference to estimate the error rate in practical applications. As modern flash memories have been demonstrated not to be immune to terrestrial radiation, it is advisable not to rely only on flash memory to store safety-critical application key data.

5.1.2 Darknet

The beam experimental results, shown in figure 5.1, allow to evaluate the realistic radiation-induced error rate of Darknet and to compare the measured sensitivities of the considered architectures. *K40* is the only tested GPU which has ECC-protected memories. For the *K40* two configurations were tested: Unhardened (i.e., without ECC or any software reliability) and with ECC enabled. The *Titan X* and *Tegra X1* were tested using Unhardened only. Figure 5.1 shows Darknet normalized FIT (the y-axes is in log scale) for the three different architectures on Darknet. Experimental data is presented with Poisson's 95% confidence intervals, and results are shown separately for SDC and Crashes.

From Figure 5.1 it is clear that the radiation sensitivity of Darknet strongly depends on the architecture. The error rate of the Unhardened version of Darknet, in fact, changes of about one order of magnitude across the considered architectures. *Titan X* is the most reliable architecture for Darknet, as its FIT rates are about $10\times$ lower than *K40*'s. The main reasons for Pascal devices to be more reliable are: (1) 3-D transistors have shown an $10\times$ reduced per bit sensitivity to neutron compared to standard planar devices (NOH et al., 2015). The raw resources corruption probability for the *Titan X* (built using FinFET technology) is then expected to be lower than that for the *K40* (built in standard CMOS technology). (2) Pascal architecture has been explicitly designed and optimized for neural network execution. *Titan X* is then likely to use its resources in a more efficient (and, eventually, reliable) way compared to the *K40*. *Tegra X1* has an error rate which is about half the one of the K40. This is mainly caused by the much smaller area of the Tegra X1 and by the different transistor layout (20nm for the *Tegra X1* and 28nm for the *K40*).

Even if crashes are more frequent than SDCs, they could be considered less critical, since crashes could be, at least, detected (LI et al., 2008; NAKKA et al., 2005; PATTABIRAMAN et al., 2006). However, crash detection in a real-time system must guarantee that deadlines are met. That is, the system must be able to recover before causing any harm to the environment (CANDEA; FOX, 2001; LEE; SHA, 2005; WU; KUO; CHANG, 2006).

5.1.3 Faster RCNN

Figure 5.1 shows that Faster RCNN has the highest FIT rate among the tested benchmarks. For *K40* Unhardened, the Faster RCNN SDC FIT rate is $4.8\times$ greater than the Darknet one, and $3.1\times$ greater than Darknet with ECC. Comparing Darknet and Faster RCNN on *Titan X*, the FIT rate difference is $120\times$. Faster RCNN has higher Crash FIT rate compared to HOG and Darknet. This is because Faster RCNN has lots of GPU/HOST interactions, due to the many layers it requires and the multiple execution pipeline, which tends to produce more crashes due to bus errors.

As Darknet, Faster RCNN also has a significant difference between SDC FIT rate vs. Crash FIT rate for *K40* when ECC is enabled. The higher Crash FIT rate when ECC is enabled is because when ECC detects a double bit flip on memory, it launches an exception to the operating system, leading to a Crash. On a first look the Faster RCNN hight SDC FIT seems to be alarming, but on Section 5.2.3 the criticality analysis will evaluate if all radiation induced errors are in fact critical.

5.2 Pedestrian detection criticality evaluation

As discussed in Chapter 4, using only SDC FIT rate it is not possible to state *a priori* if that HOG, Darknet and Faster RCNN, corruption outputs are critical or not, i.e., if detections are still fine. It is necessary to consider also BBs' position using a statistical metric to make sure that the errors are critical or not, in this work Precision and Recall metrics are considered.

5.2.1 HOG

Figure 5.2 shows the values for Precision and Recall for both *APU A10-7850K* and *Tegra K1* SoCs, obtained comparing the corrupted output with the radiation-free output. Most of the values for both Precision and Recall are concentrated on the top-right side of figure 5.2. As discussed in Chapter 4, Precision is the fraction of retrieved instances that are relevant. If Precision is lower than 100%, some detections are not relevant, i.e. radiation leads HOG to mark as a pedestrian something it would not have marked without radiation. Having a low Precision value is risky in the sense that the system may trigger the vehicle to stop without apparent reason. Recall is the fraction of pedestrians that are detected in the corrupted output. If Recall is lower than 100%, some pedestrians HOG would normally detect are not detected because of radiation corruption. Low values for Recall are critical as they could lead to undetected pedestrians and even to accidents if used in self-driving cars.

In only 24% of corrupted executions for *APU A10-7850K* and 40% for *Tegra K1* SoCs both Precision and Recall are preserved to 100%. As there may be several corrupted executions with the same values of Precision and Recall, circles are presented with a diameter directly proportional to the percentage of SDCs with that value of Precision and Recall. Bigger circles imply that a higher percentage of errors have that value of Precision and Recall. Values overlap in the single bubble for Precision = 100% and Recall = 100% in Figure 5.2. For those executions, HOG radiation corruption is not critical, as the error was insufficient to modify any BB in a way that refutes Equation 4.3. T_J in Equation 4.3 is an arbitrary value. Results presented here (for all tested benchmarks) are obtained with $T_J = 0.5$, which derives from detection quality evaluations presented in (FAWCETT, 2006). Higher values for T_J are likely to increase the number of radiation errors identified as critical.

Figure 5.2: HOG Precision and Recall for *APU A10-7850K* and *Tegra K1*. Only corrupted outputs are considered.

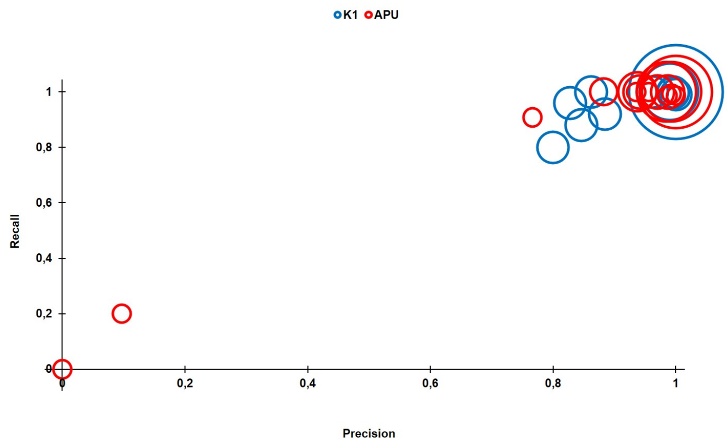


Figure 5.2 shows that some rare errors cause *APU A10-7850K* Precision and Recall to be lower than 20% or even close to 0%. Those events are extremely critical as the corrupted output has a little (if no) correlation with the expected value. As shown in Figure 5.2,most radiation corruptions for both *APU A10-7850K* and *Tegra K1* SoCs show both Precision and Recall higher than 90%. However, it is feasible to think that Precision can be relaxed, Recall should be maintained at 100%. In fact, a non-relevant detection (i.e. $Precision < 100\%$) may at most cause an unnecessary stop while a miss-detection (i.e. $Recall < 100\%$) could lead to collision and eventual human life risk. Accepting values of $Precision \geq 90\%$ but still requesting $Recall = 100\%$, 85.1% of *APU A10-7850K* and 65% of *Tegra K1* corrupted executions would not be marked as critical. Under these circumstances both *APU A10-7850K* and *Tegra K1* SDC FIT rate would be significantly reduced, eventually being sufficient to be compliant with ASIL-D.

As expected, experimental results represented in Figure 5.2 shows that radiation-induced errors are more likely to affect Precision than Recall. In fact, HOG corruption is more likely to increase the number of BB than reducing it. Additional BBs will affect Precision more than Recall.

5.2.2 Darknet

Figures 5.3a and 5.3b show the Precision and Recall for all the observed SDCs in the *K40*, *Titan X*, and *Tegra X1* obtained comparing the radiation-corrupted output with the expected output. Precision values are represented on the x-axis, and Recall values are represented on the y-axis. As mentioned earlier, if Precision is lower than 100%, some objects were wrongly detected by the Neural Network, while if Recall is lower than 100%, some objects were missed. Any Precision and Recall values lower than 100% are potentially critical.

Most errors have both Precision and Recall equal to 1 in Figure 5.3a. When the detections are calculated, all operations are made using float point numbers, so the final output of Darknet is a group of floats that represent the object coordinates. To represent the real image coordinates, it is necessary to convert the float numbers to integers. Converting float numbers to integer is a cast operation, which truncates the float number based on its exponent. Since some errors happen on last decimal places, they do not do not change precision and recall values, (i.e. precision and recall equal 1). As such, those outputs are to be considered not critical as they do not impact detection quality. So an error threshold was used to select only SDCs which change the float output values significantly. The considered error threshold was 5×10^{-3}, obtained from previous validation tests. This value was chosen because it was a good tread-off to not considering small errors which are only caused by a float representation.

Figure 5.3a shows that Precision and Recall values strongly depend on the architecture. Despite the fact that all architectures produced at least one SDC with precision and recall equal to zero, Precision and Recall patterns are different on all tested devices. From our results, it seems that the *K40* and the *Titan X* follow a similar trend, having most errors with Precision and Recall equal to 100% and then gradually reducing both Precision and Recall equally, until reaching 0%. Then, some errors preserve Precision to 100% and gradually reduce Recall. It is worth noting that having Recall lower than 100% is to be considered more critical than having Precision lower than 100%.

The *Tegra X1*, unlike the *K40* and the *Titan X*, has most of its errors being either not critical (both Precision and Recall equal to 100%) or extremely critical (Precision and Recall equal to 0%). This is probably due to the small embedded architecture of the Tegra X1, which requires the same hardware to perform several operations on the same layer. A critical error is then likely to impact the detection significantly.

Figure 5.3b shows the differences on two *K40* configurations: unhardened and ECC. Comparing Figures 5.3b and 5.3a ECC hardening technique could not avoid critical errors to happen since there are many critical errors on *K40* ECC on. ECC appears to correct only errors which are already masked by CNNs algorithms characteristics, so even if ECC's FIT rate is $10x$ smaller than ECC off version, the critical errors are not corrected by ECC.

It is reasonable to believe that an SDC on the first convolutional layers could directly impact Precision and recall values. The final part of Darknet execution is classification, performed by two fully connected layers. Fully connected neural networks are known to be fault tolerant methods due to their parallel characteristics and the intrinsic extremely low data interdependency (SEQUIN; CLAY, 1990; PROTZEL; PALUMBO; ARRAS, 1993; TCHERNEV; MULVANEY; PHATAK, 2005). However, on CNNs, the final fully connected layers are fed by the first convolutional layers, which could bring corrupt data produced by an SDC, leading to a wrong classification. Section 5.3 will explain through architectural level fault injection how the error propagation works on Darknet. Using SASSIFI fault injection and precision and recall metrics, it is possible to understand error propagation on Darknet better.

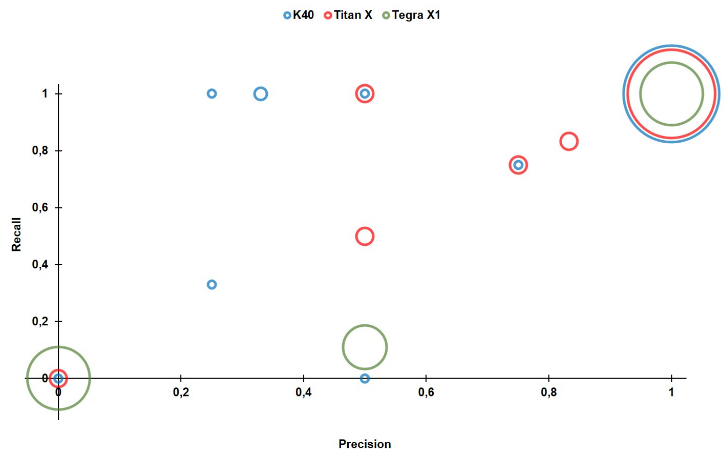
5.2.3 Faster RCNN

Faster RCNN gets the best result comparing all tested object detection algorithms on *K40* and *Titan X*. Figure 5.4a and figure 5.4b show small portions of critical errors (i.e. errors with precision $\neq$ 100% and recall $\neq$ 100%). Figure 5.4a shows the results comparing Faster RCNN on *K40* within ECC and Unhardened modes. Unlike Darknet, Faster RCNN has not many critical errors, even if it has the biggest FIT rates compared to the other algorithms. The SDCs generated by radiation could not sufficiently disrupt GPU operations to lead Faster RCCN to make bad predictions. So, a CNN raised by a combination of Python + Caffe Framework + NVIDIA cuDNN + a complex execution flow could potentially mask critical errors, once it has some overhead on memory operations and more classification steps.

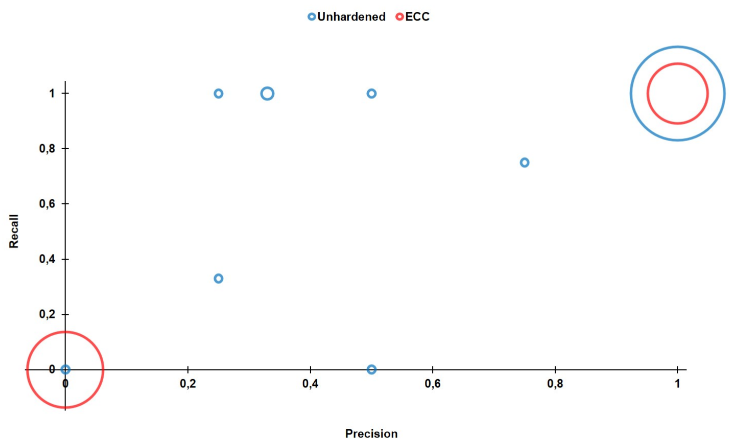
Pascal architecture used with cuDNN seems to be not only a good performance option but also ideal regarding reliability. Figure 5.4b compares Faster RCNN running on an older Kepler architecture *K40* with it running in the newest NVIDIA Pascal architecture *Titan X*. The critical errors are observed only on *K40*, *Titan X* errors are insignificant to the final output.

Figure 5.5 shows examples of radiation-induced errors for all tested applications. Figures 5.2.3 and 5.2.3 present a comparison between a gold detection and a radiation-induced error obtained on HOG on *APU A10-7850K*, this output gets values of precision and recall equals to 0%. Figures 5.2.3 and 5.2.3 show a comparison between a gold detection and a critical error obtained on Darknet on *K40*, this error gets values of 0%

Figure 5.3: Experimentally obtained values for precision and recall for all tested platforms. Values are calculated comparing corrupted outputs with the radiation-free output. 5.3a All Precision and Recall values for the tested devices *Tegra X1*, *K40* and *Titan X*. 5.3b Precision and Recall values comparing two different configurations, Unhardened and ECC for *K40*.



(a) Darknet precision and recall comparison among architectures



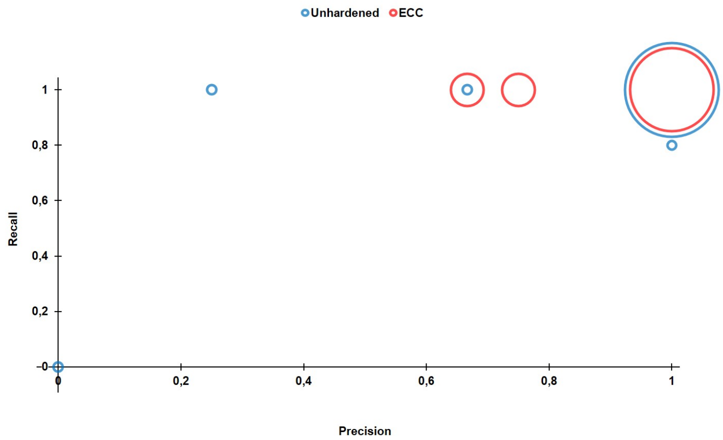
(b) Darknet precision and recall for K40

for precision and recall. Figures 5.2.3 and 5.2.3 present a comparison between a gold detection and a radiation-induced error obtained on Faster RCNN on *K40*, this error gets 100% for precision, but 80% of recall, since some objects are missing.

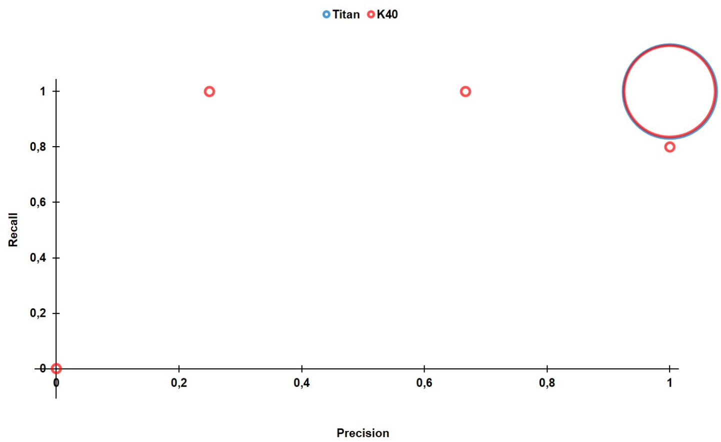
5.3 Fault Injection

This section describes the fault injection results made in this work. First HOG GDB-based fault injection results are presented. This type of fault injection serves to identify HOG critical procedures. Fault injection on Darknet is made only on open source

Figure 5.4: Experimentally obtained values for precision and recall for *K40* and *Titan X* platforms. Values are calculated comparing corrupted outputs with the radiation-free output. 5.4a Precision and Recall values comparing two different configurations, Unhardened and ECC for *K40*. 5.4b All Precision and Recall values for the tested devices *K40* and *Titan X*.

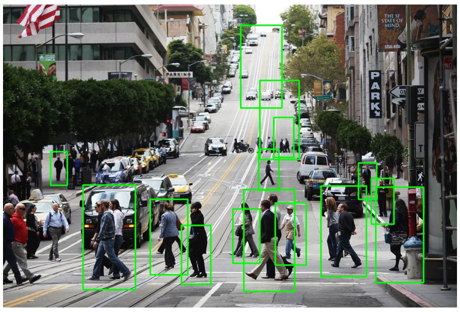


(a) Faster RCNN precision and recall comparison among architectures



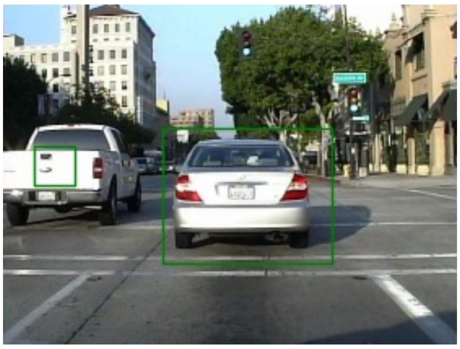
(b) Faster RCNN precision and Recall for K40

Figure 5.5: Example of experimentally observed errors on all tested applications.



(a) HOG expected output

(b) HOG corrupted output



(c) Darknet expected output

(d) Darknet corrupted output

(e) Faster RCNN expected output

(f) Faster RCNN corrupted

kernels through SASSIFI. Once matrix multiplication on Darknet is done on NVIDIA CUBLAS library, it is not possible to inject fault on those procedures, so this works is restricted to available open source kernels.

5.3.1 HOG

Following the procedures described in Chapter 4, fault injection was done on GPU kernels of HOG. The faults were injected at execution time only on the user available resources through CUDA-GDB. While impossible to be used as a comparison with radiation experiments, this data helps to understand the most critical portion of HOG. Over 2,000 faults were injected on HOG using CUDA-GDB.

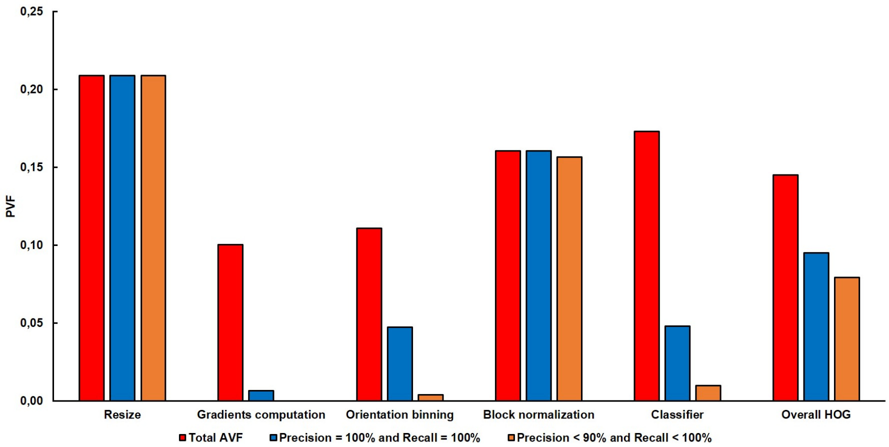
Figure 5.6 shows the results obtained with fault-injection on GPU kernels (listed and detailed in Chapter 3). For each kernel of the algorithm (and for the overall HOG), the PVF is reported, which is the percentage of injected faults that caused an observable error. As such, the PVF is an indication of how many radiation-induced errors affect computation. The higher the PVF, the more likely a radiation-induced error in the procedure is to affect HOG execution. It is worth noting that a procedure having a huge PVF does not mean that a procedure caused most radiation-induced failures observed during radiation test. In fact, PVF does not include information about the amount or sensitivity of resources involved in the computation, but only about their criticality.

As discussed in Section 5.2, the criticality evaluation is also made on SDCs produced by our CUDA-GDB fault injection, and we distinguish between critical and noncritical errors. In Figure 5.6, the PVF is presented considering only injections that reduced Precision and Recall (*Precision < 100% and Recall < 100%*) and that cause Precision to be lower than 90% and Recall to be lower than 100% (*Precision < 90% and Recall < 100%*).

The PVF for the overall HOG algorithm is 14%, which means that only 14% of the injections in all the GPU procedures impact HOG execution. Such a low PVF was expected, as HOG is an image processing code. In agreement with radiation experiment results, not all output errors impact HOG detection capability. As shown, 9% of injections cause an output error that affects Precision and Recall (Precision < 100% and Recall < 100% in Figure 5.6) and 8% of injections reduce Precision preserving Recall (Precision < 90% and Recall < 100% in Figure 5.6). Fault-injection confirms the trend observed with radiation experiment. However, for fault-injection, the portion of noncritical errors is lower than for radiation experiments. This is because our injection is performed at a high level of abstraction.

As showed in Figure 5.6, there are some portions of HOG that, once corrupted, are prone to generate critical SDCs. The most critical kernels in HOG are Resize, Block normalization, and Classifier. 20% of faults injected in *Resize* caused an SDC. Unfortunately, all the observed SDCs caused by Resize corruption are to be considered critical, as both Precision and Recall were always lower than 85%. Interestingly, high-level fault injection shows that all the SDCs in Resize were caused by faults injected in the color texture variable. Injections in the other variables of Resize did not produce any effect on HOG output. This result will be used in the next section to efficiently harden Resize and

Figure 5.6: CUDA-GDB Fault-injection PVF for HOG and its kernels.



evaluate it on SASSIFI fault injection.

Gradient Computation shows the lowest PVF and the most significant behavior. Results show that only 0.6% of SDCs could not satisfy the condition of having both Precision and Recall equal to 100%, and no injection caused Precision to be lower than 90% with a Recall lower than 100% (i.e., the PVF is 0). The Gradient Computation is a discrete derivative mask applied pixel-by-pixel to the frame. Even if a bit-flip happens while calculating the gradient on a pixel, all neighbor pixels still have the correct value, and little to no effect is expected in the output.

Orientation Binning is in charge of calculating the cells and their orientation. According to figure 5.6, Orientation Binning is very robust as its PVF is about 11%. Only 4.7% of errors injected in Orientation Binning generate SDCs that make both Precision and Recall to be lower than 100%. Relaxing criticality to Precision $<$ 90% and Recall $<$ 100%, this percentage is reduced to 0.4%. Most observed errors in this step are caused by injections that affect the cell dimensions computation, while errors in the weighted vote and the gradient calculation hardly affected the output.

The most critical GPU kernel is *Block normalization*. Corruption in this kernel results in a faulty grouping of cells or a wrong contrast normalization. The subsequent phases of HOG will then work on wrongly grouped or wrongly normalized blocks, which leads to misdetection. Block Normalization is then to be considered as a critical procedure. As shown in Figure 5.6, the PVF of Block Normalization is almost the same even if criticality is considered.

Classifier is a complex kernel. It is in charge of identifying pedestrians among objects. Classifier's PVF is 17%, and most of the SDCs produced by injection on Classifier are not critical. In fact, the PVF for Precision $<$ 90% and Recall $<$ 100% is only 0.9%.

However, some errors show an extremely low Precision (lower than 22%) and should then be carefully treated.

5.3.1.1 HOG Reliability

The experimental results reported here confirmed that HOG, being a filter and processing images, is more prone to experience Crashes than SDCs. A Crash is an undesired event in safety-critical systems but, being detectable with a watchdog, it can also be easily handled. The only constraint for Crashes hardening in a real-time system, like pedestrian detection in automotive applications, is to guarantee that deadlines are still met. In other words, even if a Crash occurs it is essential for HOG to become operative in time to maintain FTTI[1] sufficiently low to ensure safety.

SDCs, by their nature, are very hard to be detected and could be more harmful than Crashes. In the automotive market, full hardware redundancy is to be considered too expensive. As HOG is to be included in a real-time system, full software redundancy is also to be considered impractical due to its large overhead (MITRA, 2010). Based on Section 5.3.1 analysis, which goes through GPU fault injection, it is possible to identify the critical HOG procedures to be hardened, improving HOG's reliability significantly and in an efficient way. There are several ways to harden a code or portions of a code (MITRA, 2010). The scope of this section is to identify where hardening strategies should be applied, not to propose a specific hardening solution. However, to give a reference and evaluate how the fault-injection results impact the hardening solution overhead, duplication with comparison was applied, as it has already been demonstrated to detect more than 90% of SDCs in GPUs (OLIVEIRA et al., 2016). ECC reduces about one order of magnitude the SDCs rate of GPUs (OLIVEIRA et al., 2016) and, as this section shows, if available it could significantly reduce duplication overhead. In fact, when ECC is present there is no need to duplicate and check memory values but only operations and computations. In the following discussion, the overhead of duplication applied to critical HOG procedures is evaluated with or without ECC available. Table 5.1 reports the overhead imposed to HOG by the hardening (by duplication) of each phase. The overhead is higher when ECC is not present because of memory checks, which are very time-consuming.

On the fault-injection campaign, *Resize* is identified as a critical procedure. However, the observed errors were caused only by corruptions in the color parameters vari-

[1]The Fault Tolerant Time Interval (FTTI), defined (but not quantified) in the ISO26262 as the time between the fault occurrence and the action execution

Table 5.1: Execution time and selective duplication overheads for each HOG phase. Hardening values are the overhead imposed by duplication applied only to one HOG phase in relation to the unhardened version total execution time. The - symbol indicates that no hardening was needed.

Phase	Execution time percentage	Hardening overhead without ECC	Hardening overhead with ECC
Resize	5.56%	0%	0%
Gradient Computation	9.78%	-	-
Orientation Binning	47.81%	-	-
Block Normalization	14.43%	23.2%	3.2%
Classifier	22.42%	49.6%	8.6%
Total	100%	72.8%	11.8%

ables. *Resize* takes only 5.56% of overall execution time. It is not necessary to duplicate the whole procedure, but only the color texture variable. The duplication of this single variable introduces a negligible overhead in the overall HOG execution time yet significantly improves its reliability.

Gradient Computation and *Orientation binning* represents almost 9.78% and 47.81% of overall execution time, respectively. Since these two kernels did not produce critical errors, their hardening is likely not to improve reliability significantly. Based on Section 5.3.1 analysis and hardening efforts, it is not recommended hardening Gradient Computation and Orientation binning. Considering that these two steps are responsible for almost 60% of all HOG processing time, the overhead introduced by a fault tolerance technique in Gradient Computation and Orientation binning would unnecessary increase HOG execution time and power consumption. In particular, the duplication of matrices used both as input and output in Gradient Computation and Orientation binning would be extremely time-consuming, resulting in a 40.5% (5.4% with ECC) and 71% (12% with ECC) overhead, respectively. These values are not reported in Table 5.1 as, considering the overheads and the fact that these kernels produced no critical SDCs during our fault-injection campaign, it is prudent that duplication should not be applied.

Block Normalization has been identified to be the most critical kernel. Based on fault-injection campaign and hardening efforts, hardening on Block Normalization would be critical to HOG. Since this step takes only 14.43% of the overall execution time and almost all produced SDCS are critical, hardening is mandatory. An overhead of 23.2% was observed in HOG execution time when duplicating Block Normalization with no ECC protection. If ECC is available, this value drops to a much lower 3.2% as memory elements do not need to be duplicated and checked.

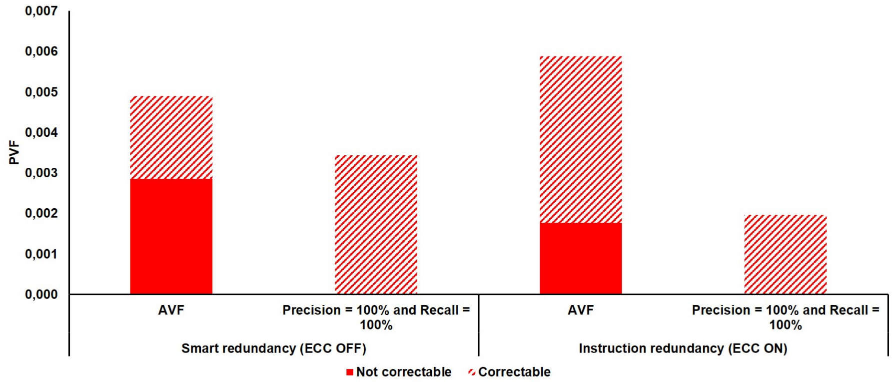
The last step of HOG execution is the *Classifier* phase. This phase got, in general, acceptable values of Precision and Recall. However, some outliers could significantly (but rarely) impact HOG reliability. Other classifiers can be used instead of the SVM-based one that was considered. It is expected from the classifier to be intrinsically a critical portion of HOG, as it decides whether an object detected by HOG is a pedestrian or not. It is then reasonable to believe that hardening the classifier is essential to ensure high reliability. Duplicating this kernel yields an overhead of 49.6% in the absence of ECC is assumed, and 8.6% if ECC is present.

A final HOG hardened version of the algorithm runs 72.8% slower if no ECC is present. It is worth noting that this overhead is considerably lower than the average slowdown imposed by a full algorithm duplication, which is about 150% (OLIVEIRA et al., 2016). As expected, ECC could significantly improve the reliability of HOG reducing the overhead imposed by duplication. In fact, as reported in Table 5.1, duplication implemented on an ECC protected device will impose an overhead of only 11.8% to HOG.

To validate the proposed hardening strategy a SASSIFI fault injection campaign was performed. As already said, SASSIFI injects fault at microinstructions level, so the fault injection models are different if compared to CUDA-GDB fault injection. While validating the hardening strategy through radiation experiments would be ideal, SASSIFI can still indicate if the GDB-based hardening is efficient to correct errors generated by a lower level of abstraction. Figure 5.7 shows the results for two HOG versions: Instruction redundancy (i.e. ECC on) and Smart Redundancy (i.e. ECC off). HOG AVF obtained with SASSIFI is extremely low, which is even more accurate than CUDA-GDB fault injection if comparing with radiation test results (see Section 5.1.1). INST and RF SASSIFI injection sites were used, but only INST injection site produced SDCs on HOG. For each version, more than 2,000 faults were injected for each error injection site resulting in a total of more than 12,000 injections.

Figure 5.7 shows that Instruction and Smart redundancy versions detected 70% and 41% of overall SDCs respectively, while 100% of critical errors were detected. However, fault injection is not accurate as radiation-induced tests, to obtain more reliable results it is necessary to expose the hardened versions of HOG to a radiation environment. Testing hardened versions of HOG will be done as a future work.

Figure 5.7: Fault-injection and AVF for HOG through SASSIFI. Three versions of HOG were tested: the same of radiation test (i.e. Unhardened), Instruction redundancy only (i.e. ECC on), and Full Redundancy (i.e. ECC off)



5.3.2 Darknet

Figure 5.8 shows the results of SASSIFI fault injection on Darknet open source kernels. It is important to emphasize that the accessible kernels represent only 13% of Darknet overall processing time, while GEMM operations consume about 67% of Darknet overall execution time. So a fault injection campaign on open source kernels will collaborate to understand if they could produce critical errors, even if they are not the time-consuming kernels of Darknet.

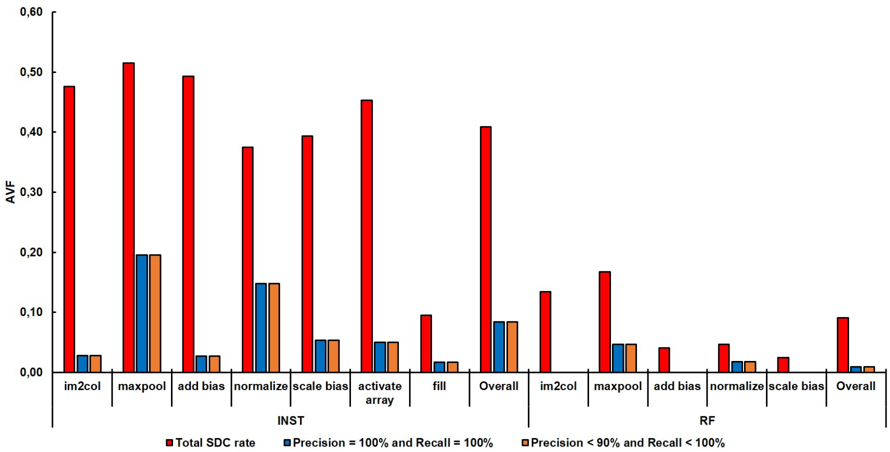
Overall AVF (figure 5.8) for INST injection site is 41%, and for for RF injection site is 9%. Relaxing criticality to Precision $< 90\%$ and Recall $< 100\%$, this percentage is reduced to less than 10% for INST and 1% for RF. Despite max pool and im2col, the other kernels are simple functions that do not stress GPU resources, so they do not have a high AVF.

Max pool layer is very reliable by its concept, once it divides a feature map into blocks and selects the biggest element of each block. So if an error corrupts a smaller element, it will not be select, so that the error will be masked.

Im2col is a data rearrangement operation which makes it possible to use matrix multiplication on CNNs, however, even if a variable is corrupted on im2col, only one feature map will be with a wrong value. So the others feature maps will still be okay, giving to CNN, in the most cases, the ability to correct predict the detection.

Based on the SASSIFI fault injection and the Darknet characteristics the obvious suggestion for an efficient hardening strategy is for GEMM functions. The next section a well know hardening strategy is proposed for CNNs, based on Darknet results.

Figure 5.8: Fault-injection and AVF for Darknet and its open source kernels.



5.3.2.1 CNNs Reliability

CNNs, in general, make an intensive use of GEMM (i.e. 67% of overall processing time), and GEMM is a sensible kernel in GPUs. Actually, most radiation-induced errors in GEMM have been demonstrated to impact multiple locations in the corrupted output (RECH et al., 2013a). Multiple errors on the first layers of a neural network could significantly reduce the reliability of the system, as the errors could propagate to the following layers.

The basic idea of ABFT strategy for neural networks is to harden the matrix multiplication kernels of Darknet using the ABFT for GPUs, it has already have been proposed in (RECH et al., 2013a). It is an extension of Huang and Abraham idea (HUANG; ABRAHAM, 1984), which is based on Freivalds' result checking approach (FREIVALDS, 1979). Input matrices A and B are coded before computation, adding column and row checksum vectors by summing all the elements in the correspondent column or row.

The result of the multiplication of the expanded matrices is a fully-checksum matrix M, where the n_{th} row and the n_{th} column contain the column (M_c) and row (M_r) checksum vectors of M, respectively (FREIVALDS, 1979). When the multiplication is finished, M column and row checksum vectors are re-calculated summing the first $n-1$ columns and $n-1$ rows of M, resulting on M'_c and M'_r, respectively. Output verification is done by comparing the checksum vectors from the multiplication and the newly computed ones.

If a mismatch is detected between $M_r[i]$ and $M'_r[i]$, it means that at least one error is present in the i_{th} row of M, and respectively for columns. If $M[i,j]$ is identified as the only error in M, it can be corrected quickly using the row (or column) checksum vectors following Equation 5.1 (HUANG; ABRAHAM, 1984).

$$M_{correct}[i, j] = M[i, j] - (M'_r[i] - M_r[i]) \quad (5.1)$$

On a GPU, the operations required to compute the checksums and detect errors can be done in $O(n)$, while error correction takes constant time (RECH et al., 2013a). The technique proposed by Huang and Abraham is only capable of correcting single output errors (HUANG; ABRAHAM, 1984), which has been experimentally demonstrated to correspond to less than 43% of the cases (RECH et al., 2013a). Thanks to experimental radiation data, the ABFT strategy was extended to correct multiple errors in the same row or column of M in constant time and randomly distributed errors in $O(e_r \times e_c)$, where e_r and e_c are the number of mismatches between M row and column checksums, respectively (details in (RECH et al., 2013a)).

6 CONCLUSIONS

In this work, the reliability of three object detection frameworks was evaluated. An experimental evaluation of representative object detection algorithms was proposed, through both fault-injection and radiation tests. Besides measuring FIT rates, the criticality of SDCs and their distribution at the output were investigated. The first experimental evaluation of HOG-based pedestrian detection reliability was proposed. Using metrics derived from the image processing community the behaviors of HOG executed on embedded GPUs exposed to atmospheric-like neutrons were investigated.

As expected, HOG is pretty robust against SDC, while it experiences many Crashes. This analysis helps to identify those errors that are critical for automotive applications, which are a small fraction of the total. While being rare, those errors could lead to accidents and thus should be carefully considered. Hight level fault-injection results highlight the most critical procedures for HOG. Those procedures are exactly the ones that should be hardened, eventually with duplications. Additionally, a microinstructions based fault injection was done to demonstrating the hardening efficiency. However, Classify hardening have shown to be useless on SASSIFI fault injection, on the contrary, Block normalization hardening was able to detect all critical errors.

In this work, the first experimental evaluation of Darknet and Faster RCNN reliability was performed. The behavior of a CNN on GPUs exposed to atmospheric-like neutrons was investigated. As most Neural Networks, Darknet and Faster RCNN are pretty robust against SDCs, while they experience a nonnegligible crash rate. This analysis helped to identify those errors that are critical to a real system application, which is a small fraction of the total. Aiming to harden the CNNs a further study at Darknet source code was done. Results show that only a small portion of Darknet is not based on matrix multiplication procedures, so hardening those methods would be worthless since their critical SDC rate was less than 10%. Based on radiation results and SASSIFI fault injection, a hardening method was proposed for Darknet, i.e., for all neural network that has a core based on matrix multiplication.

6.1 Future works

In the future, a more generic hardening strategy will be designed for object detection algorithms. Considering a frame i and a frame $i-1$, the precision and recall the difference between then will not be zero in a fault-free execution. However, if a SDC happens on the frame i and the precision or recall values get nearly zero, probably a SDC was detected. A possible generic hardening strategy will be a real time Precision and Recall measurement, which calculates the differences between the image detections.

Recently, YOLO got an upgrade in various aspects, which could achieve 40 frames per second and 78% of average accuracy. Such type of object detection must be evaluated under radiation tests and fault injection since it is the new state of object detection (REDMON; FARHADI, 2016).

Other future work is to develop an ABFT based hardening for YOLO. Actually, it is already being implemented. To demonstrate ABFT efficiency, it will be evaluated by a radiation-induced test.

The autonomous drive cars are complex AI systems, that is, to make a car driver without human help it is necessary to combine different algorithms, such as Holistic Path Planning, Sign Detection, and Free space detection. So as a future work, other parts of autonomous drive reliability will be investigated.

REFERENCES

AGENCY, E. S. **ESA COROT Mission Documentation**. 2014. Avaliable from: http://www.esa.int/Our_Activities/Space_Science/COROT. Accessed: September 2016.

AMD, A. M. D. Bios and kernel developer guide for amd family 16h models 00h-0fh processors. **Technical doc 48751 Rev 3.03**, 2015.

AMIN, S. U.; AGARWAL, K.; BEG, R. Genetic neural network based data mining in prediction of heart disease using risk factors. In: IEEE CONFERENCE ON INFORMATION COMMUNICATION TECHNOLOGIES. **Proceedings...** [S.l.]: IEEE, 2013. p. 1227–1231.

ANANDTECH. **Hot Chips 2016: NVIDIA Discloses Tegra Parker Details**. 2016. Avaliable from: http://www.anandtech.com/show/10596/hot-chips-2016-nvidia-discloses-tegra-parker-details. Accessed: September 2016.

ANGELOVA, A. et al. Real-time pedestrian detection with deep network cascades. In: PROCEEDINGS OF BMVC 2015. **Proceedings...** [S.l.]: BMVC, 2015.

ARM. **ARM Cortex-A73 MPCore Processor Technical Reference Manual Revision: r0p2**. 2016. Accessed: September 2016.

BAGATIN, M. et al. Sensitivity of nor flash memories to wide-energy spectrum neutrons during accelerated tests. In: RELIABILITY PHYSICS SYMPOSIUM, 2014 IEEE INTERNATIONAL. **Proceedings...** [S.l.]: IEEE, 2014. p. 5F.3.1–5F.3.6.

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

BECKER, J.; SANDER, O. **aramis: Project Overview**. 2013. Avaliable from: http://www.across-project.eu/workshop2013/121108_ARAMIS_Introduction_HiPEAC_WS_V3.pdf. Accessed: September 2016.

BINDER, D.; SMITH, E. C.; HOLMAN, A. B. Satellite anomalies from galactic cosmic rays. **IEEE Transactions on Nuclear Science**, v. 22, n. 6, p. 2675–2680, Dec 1975.

BLANCHET, B. et al. A static analyzer for large safety-critical software. In: PROCEEDINGS OF THE ACM SIGPLAN 2003 CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION. **Proceedings...** New York, NY, USA: ACM, 2003. (PLDI '03), p. 196–207.

BRADSKI, G. The open source computer vision (opencv) library. **Dr. Dobb's Journal of Software Tools**, v. 1, n. 1, p. 1, 2000.

BREUER, M.; GUPTA, S.; MAK, T. M. Defect and error tolerance in the presence of massive numbers of defects. **Design Test of Computers, IEEE**, v. 21, n. 3, p. 216–227, May 2004.

CANDEA, G.; FOX, A. Recursive restartability: Turning the reboot sledgehammer into a scalpel. In: PROCEEDINGS OF THE EIGHTH WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS. **Proceedings...** Washington, DC, USA: IEEE, 2001. (HOTOS '01), p. 125–.

CHETLUR, S. et al. cudnn: Efficient primitives for deep learning. **CoRR**, abs/1410.0759, 2014.

CIREGAN, D.; MEIER, U.; SCHMIDHUBER, J. Multi-column deep neural networks for image classification. In: 2012 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION. **Proceedings...** [S.l.]: IEEE, 2012. p. 3642–3649.

CIRESAN, D. C. et al. Deep neural networks segment neuronal membranes in electron microscopy images. In: PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS. **Proceedings...** USA: IEEE, 2012. (NIPS'12), p. 2843–2851.

CONSTANTINESCU, C. Impact of deep submicron technology on dependability of vlsi circuits. In: DEPENDABLE SYSTEMS AND NETWORKS, 2002. DSN 2002. PROCEEDINGS. INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2002. p. 205–209.

CUMMINGS, D. M. Embedded software under the courtroom microscope: A case study of the toyota unintended acceleration trial. **IEEE Technology and Society Magazine**, v. 35, n. 4, p. 76–84, Dec 2016.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: COMPUTER VISION AND PATTERN RECOGNITION, 2005. CVPR 2005. IEEE COMPUTER SOCIETY CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2005. v. 1, p. 886–893 vol. 1.

DEBARDELEBEN, N. et al. GPU Behavior on a Large HPC Cluster. **6th Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids in conjunction with the 19th International European Conference on Parallel and Distributed Computing (Euro-Par 2013), Aachen, Germany,**, August 26-30 2013.

DICELLO, J.; PACIOTTI, M.; SCHILLACI, M. An estimate of error rates in integrated circuits at aircraft altitudes and at sea level. **Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms**, v. 40, p. 1295 – 1299, 1989.

DOLLAR, P. et al. Pedestrian detection: A benchmark. In: 2009 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION. **Proceedings...** [S.l.]: IEEE, 2009. p. 304–311.

DOLLAR, P. et al. Pedestrian detection: An evaluation of the state of the art. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 34, n. 4, p. 743–761, April 2012.

DONGARRA, J.; MEUER, H.; STROHMAIER, E. **National Highway Traffic Safety Administration report**. 2013. Avaliable from: https://www.nhtsa.gov. Accessed: September 2016.

DONGARRA, J.; MEUER, H.; STROHMAIER, E. **ISO26262 Standard**. 2015. Avaliable from: https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-1:v1:en. Accessed: September 2016.

EL-SAYED, N.; SCHROEDER, B. Reading between the lines of failure logs: Understanding how hpc systems fail, DSN. In: DSN. **Proceedings...** [S.l.]: IEEE, 2013.

ESSEN, B. V. et al. Lbann: Livermore big artificial neural network hpc toolkit. In: PROCEEDINGS OF THE WORKSHOP ON MACHINE LEARNING IN HIGH-PERFORMANCE COMPUTING ENVIRONMENTS. **Proceedings...** New York, NY, USA: IEEE, 2015. (MLHPC '15), p. 5:1–5:6.

EVERINGHAM, M. et al. The pascal visual object classes (voc) challenge. **International Journal of Computer Vision**, v. 88, n. 2, p. 303–338, 2010.

FANG, B. et al. Gpu-qin: A methodology for evaluating the error resilience of gpgpu applications. In: TO APPEAR IN THE PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE (ISPASS). **Proceedings...** [S.l.]: IEEE, 2014.

FANG, B. et al. Gpu-qin: A methodology for evaluating the error resilience of gpgpu applications. In: PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE (ISPASS), 2014 IEEE INTERNATIONAL SYMPOSIUM ON. **Proceedings...** [S.l.]: IEEE, 2014. p. 221–230.

FANG, B. et al. Poster: Evaluating error resiliency of gpgpu applications. In: HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS (SCC), 2012 SC COMPANION:. **Proceedings...** [S.l.]: IEEE, 2012. p. 1504–1504.

FAWCETT, T. An introduction to roc analysis. **Pattern recognition letters**, Elsevier, v. 27, n. 8, p. 861–874, 2006.

FOGLE, A. D. et al. Flash memory under cosmic and alpha irradiation. **IEEE Transactions on Device and Materials Reliability**, v. 4, n. 3, p. 371–376, Sept 2004.

FREIVALDS, R. Fast probabilistic algorithms. In: BECVAR, J. (Ed.). **Mathematical Foundations of Computer Science 1979**. [S.l.]: Springer Berlin Heidelberg, 1979, (Lecture Notes in Computer Science, v. 74). p. 57–69.

GOLDHAGEN, P. Cosmic-ray neutrons on the ground and in the atmosphere. **MRS bulletin**, Cambridge Univ Press, v. 28, n. 02, p. 131–135, 2003.

GOMEZ, L. A. B. et al. Gpgpus: How to combine high computational power with high reliability. In: **2014 Design Automation and Test in Europe Conference and Exhibition**. Dresden, Germany: [s.n.], 2014.

GOOGLE. **Google Self-Driving Car Project Montly Report**. 2016. Avaliable from: https://www.google.com/selfdrivingcar/reports/. Accessed: September 2016.

GOOGLE. **Wired Google's Self-Driving Car Caused Its First Crash**. 2016. Avaliable from: http://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/. Accessed: September 2016.

HAQUE, I.; PANDE, V. Hard data on soft errors: A large-scale assessment of real-world error rates in gpgpu. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2010 10TH IEEE/ACM INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: ACM, 2010. p. 691–696.

HARI, S. et al. Tganges: Gang error simulation for hardware resiliency evaluation. In: PROCEEDING OF THE 41ST ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECUTURE. **Proceedings...** [S.l.]: IEEE, 2014. (ISCA '14).

HARI, S. K. S. et al. Sassifi: Evaluating resilience of gpu applications. In: PROCEEDINGS OF THE WORKSHOP ON SILICON ERRORS IN LOGIC-SYSTEM EFFECTS (SELSE). **Proceedings...** [S.l.]: IEEE, 2015.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture, Fifth Edition: A Quantitative Approach**. 5th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

HERRERA-ALZU, I.; LOPEZ-VALLEJO, M. System design framework and methodology for xilinx virtex fpga configuration scrubbers. **Nuclear Science, IEEE Transactions on**, v. 61, n. 1, p. 619–629, Feb 2014.

HUANG, K.-H.; ABRAHAM, J. Algorithm-based fault tolerance for matrix operations. **Computers, IEEE Transactions on**, C-33, n. 6, p. 518–528, June 1984.

HWANG, A. A.; STEFANOVICI, I. A.; SCHROEDER, B. Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design. **ACM SIGPLAN Notices**, ACM, v. 47, n. 4, p. 111–122, 2012.

INTEL. **Intel Xeon Phi Coprocessor System Software Developers Guide**. 2014. Accessed: September 2016.

INTEL. Intel 64 and ia-32 architectures software developers manual combined volumes: 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c and 3d. 2016.

JEDEC. **Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices**. [S.l.], 2006.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. In: PROCEEDINGS OF THE 22ND ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA. **Proceedings...** New York, NY, USA: ACM, 2014. (MM '14), p. 675–678.

JOACHIMS, T. Making large-scale SVM learning practical. In: SCHÖLKOPF, B.; BURGES, C.; SMOLA, A. (Ed.). **Advances in Kernel Methods - Support Vector Learning**. Cambridge, MA: MIT Press, 1999. chp. 11, p. 169–184.

JUST, G. et al. Soft errors induced by natural radiation at ground level in floating gate flash memories. In: RELIABILITY PHYSICS SYMPOSIUM (IRPS), 2013 IEEE INTERNATIONAL. **Proceedings...** [S.l.]: IEEE, 2013. p. 3D.4.1–3D.4.8.

KANG, M.; LIM, Y. C. Pedestrian detection using hog-based block selection. In: INFORMATICS IN CONTROL, AUTOMATION AND ROBOTICS (ICINCO), 2014 11TH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2014. v. 02, p. 783–787.

KNIGHT, J. C. Safety critical systems: challenges and directions. In: SOFTWARE ENGINEERING, 2002. ICSE 2002. PROCEEDINGS OF THE 24RD INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2002. p. 547–550.

KOOPMAN, P. A case study of toyota unintended acceleration and software safety. **Presentation. Sept**, 2014.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Proceedings...** [S.l.]: IEEE, 2012. p. 2012.

LAPRIE, J. C. Dependable computing and fault tolerance : Concepts and terminology. In: FAULT-TOLERANT COMPUTING, 1995, HIGHLIGHTS FROM TWENTY-FIVE YEARS., TWENTY-FIFTH INTERNATIONAL SYMPOSIUM ON. **Proceedings...** [S.l.]: IEEE, 1995. p. 2–.

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

LEE, K.; SHA, L. Process resurrection: a fast recovery mechanism for real-time embedded systems. In: 11TH IEEE REAL TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM. **Proceedings...** [S.l.]: IEEE, 2005. p. 292–301.

LI, M.-L. et al. Understanding the propagation of hard errors to software and implications for resilient system design. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 42, n. 2, p. 265–276, mar 2008.

LI, X.; GUO, X. A hog feature and svm based method for forward vehicle detection with single camera. In: INTELLIGENT HUMAN-MACHINE SYSTEMS AND CYBERNETICS (IHMSC), 2013 5TH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2013. v. 1, p. 263–266.

LIANG, Y. et al. Bluegene/l failure analysis and prediction models. In: DEPENDABLE SYSTEMS AND NETWORKS, 2006. DSN 2006. INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2006. p. 425–434.

LIANG, Y. et al. Filtering failure logs for a bluegene/l prototype. In: DEPENDABLE SYSTEMS AND NETWORKS, 2005. DSN 2005. PROCEEDINGS. INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2005. p. 476–485.

LIM, J. J.; ZITNICK, C. L.; DOLLAR, P. Sketch tokens: A learned mid-level representation for contour and object detection. In: THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR). **Proceedings...** [S.l.]: IEEE, 2013.

LOWE, D. G. Object recognition from local scale-invariant features. In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON COMPUTER VISION-VOLUME 2 - VOLUME 2. **Proceedings...** Washington, DC, USA: IEEE, 1999. (ICCV '99), p. 1150–.

LUO, P. et al. Switchable deep network for pedestrian detection. In: 2014 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION. **Proceedings...** [S.l.]: IEEE, 2014. p. 899–906.

LUTZ, R. R. Analyzing software requirements errors in safety-critical, embedded systems. In: REQUIREMENTS ENGINEERING, 1993., PROCEEDINGS OF IEEE INTERNATIONAL SYMPOSIUM ON. **Proceedings...** [S.l.]: IEEE, 1993. p. 126–133.

MARTINO, C. D. et al. Lessons learned from the analysis of system failures at petascale: The case of blue waters. **44th international**, 2014.

MARWEDEL, P. **Embedded system design: Embedded systems foundations of cyber-physical systems**. [S.l.]: Springer Science & Business Media, 2010.

MAY, T. C.; WOODS, M. H. Alpha-particle-induced soft errors in dynamic memories. **IEEE Transactions on Electron Devices**, v. 26, n. 1, p. 2–9, Jan 1979.

MICHALAK, S. E. et al. Assessment of the impact of cosmic-ray-induced neutrons on hardware in the roadrunner supercomputer. **IEEE Transactions on Device and Materials Reliability**, v. 12, n. 2, p. 445–454, June 2012.

MITRA, S. **System-Level Single-Event Effects**. 2010. IEEE Nuclear and Space Radiation Effects Conference, NSREC 2012 Short Course. Accessed: September 2016.

MONEY, C. **Toyota settles acceleration case after $3 million jury verdict**. 2013. Http://money.cnn.com/2013/10/25/news/companies/toyota-crash-verdict/. Accessed: September 2016.

MUKHERJEE, S. S. et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: PROCEEDINGS OF THE 36TH ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE. **Proceedings...** Washington, DC, USA: ACM, 2003. p. 29–.

NAKKA, N. et al. An architectural framework for detecting process hangs/crashes. In: ____. **Dependable Computing - EDCC 5: 5th European Dependable Computing Conference, Budapest, Hungary, April 20-22, 2005. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 103–121.

NASA. **Magnet field**. 2005. Http://sec.gsfc.nasa.gov/popscise.jpg. Accessed: September 2016.

NEAGOE, V. E.; CIOTEC, A. D.; BARAR, A. P. A concurrent neural network approach to pedestrian detection in thermal imagery. In: 2012 9TH INTERNATIONAL CONFERENCE ON COMMUNICATIONS (COMM). **Proceedings...** [S.l.]: IEEE, 2012. p. 133–136.

NICOLAIDIS, M. Time redundancy based soft-error tolerance to rescue nanometer technologies. In: VLSI TEST SYMPOSIUM, 1999. PROCEEDINGS. 17TH IEEE. **Proceedings...** [S.l.]: IEEE, 1999. p. 86–94.

NOH, J. et al. Study of neutron soft error rate (ser) sensitivity: Investigation of upset mechanisms by comparative simulation of finfet and planar mosfet srams. **Nuclear Science, IEEE Transactions on**, v. 62, n. 4, p. 1642–1649, Aug 2015.

NVIDIA. **NVIDIA Jetson TK1 developer kit**. 2014. Avaliable from: http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html. Accessed: September 2016.

NVIDIA. **NVIDIA Jetson TX1 developer kit**. 2015. Avaliable from: www.nvidia.com/object/jetson-tx1-dev-kit.html. Accessed: September 2016.

NVIDIA. **NVIDIAs Next Generation CUDA Compute Architecture: Kepler GK110**. 2015. Avaliable from: http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf. Accessed: September 2016.

NVIDIA. Tegra x1 nvidia new mobile superchip. **Whitepaper**, 2015.

NVIDIA. **NVIDIA TITAN X**. 2016. Avaliable from: https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/. Accessed: September 2016.

NVIDIA. **NVIDIA PARTNER INNOVATION**. 2017. Avaliable from: www.nvidia.com/object/automotive-partner-innovation.html. Accessed: September 2016.

OLINER, A.; STEARLEY, J. What supercomputers say: A study of five system logs. In: DEPENDABLE SYSTEMS AND NETWORKS, 2007. DSN'07. 37TH ANNUAL IEEE/IFIP INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2007. p. 575–584.

OLIVEIRA, D. et al. Modern gpus radiation sensitivity evaluation and mitigation through duplication with comparison. **Nuclear Science, IEEE Transactions on**, v. 61, n. 6, p. 3115–3122, Dec 2014.

OLIVEIRA, D. A. G. et al. Gpgpus ecc efficiency and efficacy. In: INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI AND NANOTECHNOLOGY SYSTEMS (DFT 2014). **Proceedings...** [S.l.]: IEEE, 2014.

OLIVEIRA, D. A. G. de et al. Evaluation and mitigation of radiation-induced soft errors in graphics processing units. **IEEE Transactions on Computers**, v. 65, n. 3, p. 791–804, March 2016.

PAPAGEORGIOU, C. P.; OREN, M.; POGGIO, T. A general framework for object detection. In: SIXTH INTERNATIONAL CONFERENCE ON COMPUTER VISION (IEEE CAT. NO.98CH36271). **Proceedings...** [S.l.]: IEEE, 1998. p. 555–562.

PATTABIRAMAN, K. et al. Dynamic derivation of application-specific error detectors and their implementation in hardware. In: 2006 SIXTH EUROPEAN DEPENDABLE COMPUTING CONFERENCE. **Proceedings...** [S.l.]: IEEE, 2006. p. 97–108.

PECCHIA, A. et al. Improving log-based field failure data analysis of multi-node computing systems. In: DEPENDABLE SYSTEMS & NETWORKS (DSN), 2011 IEEE/IFIP 41ST INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2011. p. 97–108.

PILLA, L. et al. Memory access time and input size effects on parallel processors reliability. **Nuclear Science, IEEE Transactions on**, v. 62, n. 6, p. 2627–2634, Dec 2015.

PROGRAMME, E. N. C. A. **Euro NCAP Rating Review, Report from the Ratings Group**. 2012. Avaliable from: http://www.euroncap.com. Accessed: September 2016.

PROTZEL, P. W.; PALUMBO, D. L.; ARRAS, M. K. Performance and fault-tolerance of neural networks for optimization. **IEEE Transactions on Neural Networks**, v. 4, n. 4, p. 600–614, Jul 1993.

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

RECH, P. et al. Threads distribution effects on graphics processing units neutron sensitivity. **Nuclear Science, IEEE Transactions on**, v. 60, n. 6, p. 4220–4225, Dec 2013.

RECH, P. et al. Impact of gpus parallelism management on safety-critical and hpc applications reliability. In: **IEEE International Conference on Dependable Systems and Networks (DSN 2014)**. Atlanta, USA: [s.n.], 2014.

REDMON, J. et al. You only look once: Unified, real-time object detection. **CoRR**, abs/1506.02640, 2015.

REDMON, J.; FARHADI, A. YOLO9000: better, faster, stronger. **CoRR**, abs/1612.08242, 2016.

REN, S. et al. Faster R-CNN: Towards real-time object detection with region proposal networks. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS (NIPS). **Proceedings...** [S.l.]: IEEE, 2015.

REN, X.; RAMANAN, D. Histograms of sparse codes for object detection. In: THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR). **Proceedings...** [S.l.]: IEEE, 2013.

RIBEIRO, D. et al. A real-time pedestrian detector using deep learning for human-aware navigation. **CoRR**, abs/1607.04441, 2016.

SAGGESE, G. P. et al. An experimental study of soft errors in microprocessors. **IEEE Micro**, v. 25, n. 6, p. 30–39, Nov 2005.

SAHOO, R. K. et al. Failure data analysis of a large-scale heterogeneous server environment. In: DEPENDABLE SYSTEMS AND NETWORKS, 2004 INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2004. p. 772–781.

SAXENA, N. R. Autonomous car is the new driver for resilient computing and design-for-test. In: SILICON ERRORS IN LOGIC - SYSTEM EFFECTS (SELSE) KEYONTE. **Proceedings...** [S.l.]: IEEE, 2016.

SCHROEDER, B.; GIBSON, G. A large-scale study of failures in high-performance computing systems. **Dependable and Secure Computing, IEEE Transactions on**, IEEE, v. 7, n. 4, p. 337–350, 2010.

SCHROEDER, B.; GIBSON, G. A. Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? In: FAST. **Proceedings...** [S.l.]: IEEE, 2007. v. 7, p. 1–16.

SCHROEDER, B.; PINHEIRO, E.; WEBER, W.-D. Dram errors in the wild: A large-scale field study. In: SIGMETRICS. **Proceedings...** [S.l.]: IEEE, 2009.

SCHROEDER, B.; PINHEIRO, E.; WEBER, W.-D. Dram errors in the wild: a large-scale field study. In: ACM SIGMETRICS PERFORMANCE EVALUATION REVIEW. **Proceedings...** [S.l.]: ACM, 2009. v. 37, n. 1, p. 193–204.

SEIFERT, N.; ZHU, X.; MASSENGILL, L. W. Impact of scaling on soft-error rates in commercial microprocessors. **Nuclear Science, IEEE Transactions on**, IEEE, v. 49, n. 6, p. 3100–3106, 2002.

SEQUIN, C. H.; CLAY, R. D. Fault tolerance in artificial neural networks. In: 1990 IJCNN INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS. **Proceedings...** [S.l.]: IEEE, 1990. p. 703–708 vol.1.

SERMANET, P. et al. Overfeat: Integrated recognition, localization and detection using convolutional networks. **CoRR**, abs/1312.6229, 2013.

SIVARAMAN, S.; TRIVEDI, M. M. Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis. **IEEE Transactions on Intelligent Transportation Systems**, v. 14, n. 4, p. 1773–1795, Dec 2013.

SRIDHARAN, V.; KAELI, D. R. The effect of input data on program vulnerability. In: PROCEEDINGS OF THE 2009 WORKSHOP ON SILICON ERRORS IN LOGIC AND SYSTEM EFFECTS. **Proceedings...** [S.l.]: IEEE, 2009. (SELSE '09).

SRIDHARAN, V. et al. Feng shui of supercomputer memory: positional effects in dram and sram faults. In: PROCEEDINGS OF SC13: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS. **Proceedings...** [S.l.]: IEEE, 2013. p. 22.

STEPHENSON MARK SASTRY HARI, S. K. et al. Flexible software profiling of gpu architectures. In: PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE. **Proceedings...** [S.l.]: IEEE, 2015. (ISCA '15), p. 185–197.

STRATEGIES, S. R. **Toyota Unintended Acceleration and the Big Bowl of Spaghetti Code**. 2013. Avaliable from: www.safetyresearch.net/blog/articles/toyota-unintended-acceleration-and-big-bowl-spaghetti-code. Accessed: September 2016.

SZEGEDY, C.; TOSHEV, A.; ERHAN, D. Deep neural networks for object detection. In: BURGES, C. J. C. et al. (Ed.). **Advances in Neural Information Processing Systems 26**. [S.l.]: Curran Associates, Inc., 2013. p. 2553–2561.

TAN, J. et al. Analyzing soft-error vulnerability on gpgpu microarchitecture. In: WORKLOAD CHARACTERIZATION (IISWC), 2011 IEEE INTERNATIONAL SYMPOSIUM ON. **Proceedings...** [S.l.]: IEEE, 2011. p. 226–235.

TCHERNEV, E. B.; MULVANEY, R. G.; PHATAK, D. S. Investigating the fault tolerance of neural networks. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 17, n. 7, p. 1646–1664, jul 2005.

TIMES, T. N. Y. **Jury Finds Toyota Liable in Fatal Wreck in Oklahoma**. 2013. Avaliable from: http://www.nytimes.com/2013/10/25/business/jury-finds-toyota-liable-in-fatal-wreck-in-oklahoma.html. Accessed: September 2016.

UIJLINGS, J. R. R. et al. Selective search for object recognition. **International Journal of Computer Vision**, v. 104, n. 2, p. 154–171, 2013.

VIOLANTE, M. et al. A new hardware/software platform and a new 1/e neutron source for soft error studies: Testing fpgas at the isis facility. **Nuclear Science, IEEE Transactions on**, v. 54, n. 4, p. 1184–1189, 2007.

WOLF, M. **Computers as components: principles of embedded computing system design**. [S.l.]: Elsevier, 2012.

WOLF, M. **High-performance embedded computing: applications in cyber-physical systems and mobile computing**. [S.l.]: Newnes, 2014.

WU, C.-H.; KUO, T.-W.; CHANG, L.-P. The design of efficient initialization and crash recovery for log-based file systems over flash memory. **Trans. Storage**, ACM, New York, NY, USA, v. 2, n. 4, p. 449–467, nov 2006.

WUNDERLICH, H. J.; BRAUN, C.; HALDER, S. Efficacy and efficiency of algorithm-based fault-tolerance on gpus. In: ON-LINE TESTING SYMPOSIUM (IOLTS), 2013 IEEE 19TH INTERNATIONAL. **Proceedings...** [S.l.]: IEEE, 2013. p. 240–243.

YANG, X.; ZHANG, C.; TIAN, Y. Recognizing actions using depth motion maps-based histograms of oriented gradients. In: PROCEEDINGS OF THE 20TH ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA. **Proceedings...** New York, NY, USA: ACM, 2012. (MM ’12), p. 1057–1060.

YEN, T.-Y.; WOLF, W. **Hardware-software co-synthesis of distributed embedded systems**. [S.l.]: Springer Science & Business Media, 2013.

ZHU, Q. et al. Fast human detection using a cascade of histograms of oriented gradients. In: COMPUTER VISION AND PATTERN RECOGNITION, 2006 IEEE COMPUTER SOCIETY CONFERENCE ON. **Proceedings...** [S.l.]: IEEE, 2006. v. 2, p. 1491–1498.