Evaluating Convolutional Neural Networks Reliability depending on their Data Representation

Annachiara Ruospo*, Alberto Bosio†, Alessandro Ianne*, Ernesto Sanchez*
*Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy
†INL, Ecole Centrale de Lyon, Lyon, France
{annachiara.ruospo, alessandro.ianne, ernesto.sanchez}@polito.it
alberto.bosio@ec-lyon.fr

Abstract—Safety-critical applications are frequently based on deep learning algorithms. In particular, Convolutional Neural Networks (CNNs) are commonly deployed in autonomous driving applications to fulfil complex tasks such as object recognition and image classification. Ensuring the reliability of CNNs is thus becoming an urgent requirement since they constantly behave in human environments. A common and recent trend is to replace the full-precision CNNs to make way for more optimized models exploiting approximation paradigms such as reduced bit-width data type. If from one hand this is poised to become a sound solution for reducing the memory footprint as well as the computing requirements, it may negatively affect the CNNs resilience. The intent of this work is to assess the reliability of a CNN-based system when reduced bit-widths are used for the network parameters (i.e., synaptic weights). The approach evaluates the impact of permanent faults in CNNs by adopting several bit-width schemes and data types, i.e., floating-point and fixed-point. This determines the trade-off between the CNN accuracy and the bits required to represent network weights. The characterization is performed through a fault injection environment built on the *darknet* open source framework. Experimental results show the effects of permanent fault injections on the weights of LeNet-5 CNN.

Index Terms—Deep Learning, Test, Reliability, Fault Injection, Safety, Automotive

1 Introduction

Deep Learning [1] is currently one of the most intensively and widely used predictive model in the field of machine learning. In this light, Convolutional Neural Networks (CNNs) are gaining popularity due to their excellent performance in solving complex learning problems [2]. Indeed, they provide very good results for many tasks such as object recognition in images/videos, drug discovery, natural language processing up to playing games [3]–[5]. CNNs are a subset of Deep Neural Networks (DNNs) and include multiple layers: convolutional, non-linearity, pooling and fully-connected. Their name origins from the mathematical linear operation between matrixes called convolution.

Neural networks may be considered robust, from a theoretical perspective, due to their iterative nature and learning process [6]. However, especially when deployed in *safety-critical applications* such as autonomous driving [7], their resilience must be evaluated. More in detail, the probability that a hardware or software fault may cause a system failure.

As a rule, the reliability analysis for electronic devices is regulated by standards which depend on the application domain (e.g., IEC 61508 for industrial systems, DO-254 for avionics, ISO 26262 for automotive) [8]. Since CNNs are increasingly deployed in automotive application, it is necessary to figure out how to map the automotive standards requirements, i.e., ISO 26262, to the deep learning systems. A first attempt is the reliability assessment through fault injection campaigns. Usually, to improve the reliability of electronic devices in the autonomous domain, in-field test solutions are embedded and activated in mission-mode to detect possible permanent faults before these may produce any failure. Examples of such test solutions are Design for Testability techniques (e.g., BIST), self-test functional approaches (e.g, Software-based Self-test [9]), or a combination of both. Independently on the adopted test solution, the key point is the achieved fault coverage with respect to the adopted fault model(s). A higher fault coverage leads to ensure a higher level of safety.

Recent trends in Convolutional Neural Network research area show a growing adoption of more optimized models that use a reduced bit-width data type in either training or inference phase. Indeed, one important limitation about the adoption of the newer version of CNNs is the memory required for storing the network parameters. For instance, VGG-Net [10], one of the most deeper implementation, requires 500 MB of memory, a value that goes outside the possibility of many constrained hardware platforms. In the last few years, Approximate Computing (AxC) has become a major field of research to improve both speed and energy consumption in embedded and high-performance systems [11]. By relaxing the need for fully precise or completely deterministic operations, approximate computing substantially improves energy efficiency and reduces the memory requirement. Various techniques for approximate comput-

978-1-7281-9535-3/20/$31.00 ©2020 IEEE
DOI 10.1109/DSD51259.2020.00109

ing augment the design space by providing another set of design knobs for performance-accuracy trade-offs. As an example, the gain in energy between a low-precision 8-bit operation suitable for vision and a 64-bit double-precision floating-point operation necessary for high-precision scientific computation can reach up to 50× by considering storage, transport and computation. The gain in energy efficiency (the number of computations per Joule) is even larger since the delay of basic operations is greatly reduced. Having simpler operators also reduces implementation cost, which allows the network to use more resources in parallel.

Borrowing these ideas from the Approximate Computing field, the major challenge is to find an adequate data representation for CNNs that fits well with the application and hardware constraints. CNNs lend themselves well to Approximate Computing techniques, especially with fixed-point arithmetic or low-precision floating-point implementations, which exposes large fine-grain parallelism. For instance, in [12] the authors describe a *binary network* which exploits only two values $\{-1, 1\}$ for the weights representation. Another proposed solution is the *ternary network* [13]; it quantizes weights into 3 different values $\{-1, 0, 1\}$. Finally, XNOR-Net [14] uses a slightly different methodology: all the computations are performed thought XNOR and bit counting operations, at the same time reducing the precision of all the operands involved during the computation.

At this point the question might sound trivial: are those optimized models reliable enough to tolerate failures that propagate throughout the system? It starts to be necessary to evaluate CNNs behaviour in a faulty scenario to determine if they can still be safely deployed in a safety-critical system. These doubts are justified if considering the growing technology scaling in chip manufacturing. Due to the transistors shrinking, newer hardware platforms are more complex and at the same time more susceptible to faults, albeit faster.

The end-goal of this paper is to characterize the impact of permanent faults affecting a Convolutional Neural Network by means of fault injection campaigns, when a more compact representation is used for describing the network parameters. We analysed different implementations of the same CNN architecture, when different data types are exploited. Moreover, this work aims at study the criticality level of the network layers as well as identifying all the *Safe Faults Application Dependent* (SFAD), those faults that do not produce any failure in the operational mode (ISO 26262).

The rest of the paper is organized as follows. The state-of-the-art research work and the main contributions are highlighted in Section 1.1. Section 2 presents the case study, focusing on the experiment set-up and the weight conversion technique. Section 3 describes the fault injection framework, whereas Section 4 provides the experimental results. Finally, Section 5 concludes the article by outlining some of the possible future research directions.

1.1 Related Work

In literature, more and more attention is paid to the Neural Network Reliability. Depending on multiple factors, such as fault injection typology, level of abstraction, fault models, it is possible to identify different sets of interesting research activities.

A significant set focuses on analysing a specific fault model: the *soft* error (i.e., bit flip). In [15], the authors evaluate the reliability of one CNN executed on three different GPU architectures (Kepler, Maxwell, and Pascal). The soft errors injection has been done by exposing the GPUs running the CNN under controlled neutron beams. A similar but wider approach is detailed in [16], where the authors assess the reliability of a 54-layers DNN (NVIDIA DriveWorks) through fault injection experiments for permanent faults and accelerated neutron beam testing for transient errors. Faults are injected on the network weights and on the input images. All the inferences are executed on Volta GPU only targeting floating point values. Moving forward, the authors present in [17] a different analysis. They characterize the propagation of soft errors from the hardware to the application software of different CNNs. The injections are performed by using a Deep Neural Network simulator based on open-source simulator framework, Tiny-CNN [18]. Thanks to the flexibility of the simulator, it is possible to characterize each layer for a more precise analysis. A different framework is shown in [19]: Ares, a light-weight Deep Neural Network fault injection framework. The authors present an empirical study on the resilience of three prominent types of DNNs (fully connected, CNNs and Gated Recurrent Unit). In particular, they focus on two fixed-point data types for each network: $Q_{3,13}$ i.e, 3 integer and 13 fractional bits, and $Q_{2,6}$. Their experiments demonstrate that the optimized $Q_{2,6}$ data type is 10x more fault tolerant. The reason lies in the fact that the unnecessary larger range of integer values increases the chance of failures happening. It is worth noting that this result is in line with our gathered results, presented in Section 4.2. It is a common trend to explore fixed-point computations for ultra-low power embedded systems with a limited power budget [20]. Finally, in [21], the authors analyse the reliability of a Deep Neural Network accelerator by following a High Level Synthesis (HLS) approach. They characterize the effects of both permanent and transient faults by exploiting a fault injector framework embedded into the RTL design of the accelerator. Faults are injected during the inference cycles only on a sub-set of registers: those that are in charge to store weights, input values and intermediate ones used throughout the inference job, without considering the effects of faults in the other data-path units. Regarding the data representation, they perform the experiments by only adopting a 16-bits fixed-point low precision model, claiming a negligible accuracy loss with respect to a full-precision data model.

The main contribution of this paper is a comprehensive analysis on the behavior of the CNNs depending on their data representation. In a previous work [22], we evaluated the impact of permanent faults affecting CNNs through software fault injection campaigns. However, only a floating point representation has been considered. Compared to the state-of-the-art analyses [19] [21], a wider spectrum of fixed-point representations is given (five typologies ranging from 32-bits up to 6-bits). Toward this goal, permanent faults are injected in relation with the corrupted layer and the data type.

2 Case Study

This section is intended to present the case study. As previously stated, we exploit the *darknet* open source DNN framework [23]. Implemented in C and CUDA language, it is suited perform end-to-end deployment of neural network architectures in a very simple way. It further supplies a very simple environment where several configurations of Deep Neural Networks can be executed either to perform training or inference jobs. As stated before, the work focuses on a category of DNNs, named Convolutional Neural Networks (CNNs). These artificial networks are widely exploited in fields such as image classification and object detection.

Among all the possible existing CNNs, our interest falls on LeNet-5 [24], a well-known classifier for handwritten digit recognition task introduced by Y. Lecun et al in 1998. The network architecture is composed of 1 input layer, 5 hidden layers and 1 output layer, whose typology ranges from Convolutional, Fully-Connected and Max-Pool layers. For the network reliability assessment, the behaviour of Max-Pool layer category is out of the scope of our analysis due to the fact that no arithmetic operations are implemented.

For running the experiments, the MNIST database [25], a well-known dataset used to evaluate the accuracy of new emerging models, has been selected. It is composed of 60,000 images for training and 10,000 for test/validation of the model, encoded in 28×28 pixels in grayscale. However, to lower the computational costs and time, a workload of 2023 images was randomly selected from the MNIST test/validation set for all the experiments. Moreover, since we are focusing on the inference phase and on the response of the network in a faulty scenario, a set of pre-trained weights has been adopted. It is available from the *darknet* website and includes all the weights in 32-bit floating-point. Figure 1 highlights the values distribution of these pre-trained weights. As evidenced, all the values are in the range -0.6 to 0.6 with the most of them around zero.

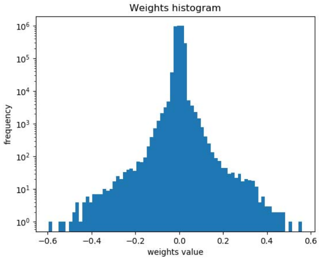


Fig. 1: Distribution of Pre-Trained Weights Values.

The reliability of the above-mentioned CNN has been evaluated by running many faults injection experiments with two different data types. The former set of experiments considers the target CNN with 32-bit floating-point weights, while the latter exploits a fixed-point representation with a bit-width ranging from 32 up to 6 bits. The next subsection (2.1) will deepen the methodology followed to switch from floating-point to fixed-point numbers, i.e., the *conversion*. Then, the Section 3 will lay out the two fault injection environments.

2.1 Weights Conversion

In neural networks field, a common approach for reducing the bit-width of weights and activation's values is to adopt quantization schemes [26] [27]. However, the *darknet* framework does not support operations with fixed-point, forcing the user to run inferences only with floating-point numbers. For this reason, it turned out to be more convenient to perform an *on-line conversion* of the network weights. This led us to slightly modify the source code to support the inferences with a lower-precision representation. All the conversions between floating-point and fixed-point have been carried out by integrating an open source library into the *darknet* framework: the *libfixmath* library [28].
Since the end-goal is to characterize faults propagation through the network (speeding up computations and compacting the model size are out of the scope of this work), we performed these on-line conversions while maintaining all the internal operations in floating point (Figure 2). The benefits coming from this approach are two-fold: first, it is not necessary to change the framework structure every time new experiments with a different data type have to be performed; second, it allows to change the representation without re-training the CNN model for each data type, exploiting the same set of trained parameters. In this way, the assessment of the CNN reliability is quicker; it is possible to switch between experiments with different numerical format in a reasonable amount of time.

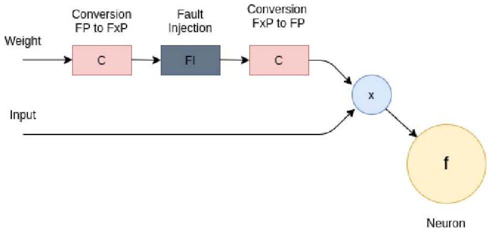


Fig. 2: On-line Weights Conversions.

For the sake of completeness, we describe how the on-line conversion of a floating-point weight is applied before reaching a single neuron (Figure 2). The applied scheme works in the following way:

1) The weight is converted from floating-point to a previously selected fixed-point representation.
2) The fixed-point weight is corrupted according to a chosen fault list and a fault location, i.e., the fault is injected.
3) The fixed-point weight is converted back to the floating-point representation in order to preserve the native implementation of the framework. In such a way, the gained value reflects the same fixed-point corrupted value, while still remaining a floating-point data.
4) The weight is multiplied by the input value.
5) The neuron performs the arithmetic computations.

Although all the network operations are executed between floating-point variables, it should be outlined that the

Scenario	Data type	Bit-width	Fractional bits	[%] Accuracy Loss
A	fixed-point	32	16	0
B	fixed-point	18	16	0
C	fixed-point	16	8	0.18
D	fixed-point	10	8	0
E	fixed-point	6	4	2.96

TABLE 1: [%] Fixed-Point LeNet-5 Accuracy Loss.

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

Listing 1: Fault Injection Pseudo-Code

loss of precision caused by the first conversion is preserved. Indeed, when moving from a high-precision representation (i.e., floating point) to a low-precision one (i.e., fixed-point), we are witnessing a truncation error effect. Then, converting back from a narrow range of value (fixed-point) to a wider one (floating-point), the truncation error still remains. This is important to justify the conversion methodology choice. Moreover, we computed the accuracy loss of the network resulting from the adoption of fixed-point weights. As highlighted in Table 1, five different scenarios have been analyzed. The second column of the Table underlines the total weight bit-width, always including 1 bit for the sign. The third column shows the amount of bits allocated to the fractional part after the radix point. The remaining bits are used for representing the integer part. To compute the accuracy of the network when weights are represented in different bit-width formats, the inference of all the images belonging to validation set of the MNIST database (10,000) have been run on LeNet-5, clearly without injecting any faults, i.e., in a golden scenario. It turned out that the accuracy loss is zero in the 60% of the cases, and lower that the 3% in the remaining 40%.

3 Fault Injection

The intent of the section is to first describe the Fault Injection environment built on the *darknet* framework. Then, more details are provided for the two case studies: the floating-point and the fixed-point weights representations.

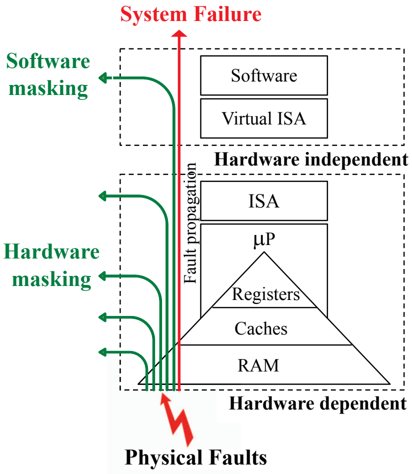


Fig. 3: System Layers and Fault Propagation.

The hardware system can be affected by faults caused by physical manufacturing defects. As Figure 3 highlights, faults could propagate through the different hardware structures composing the full system. However, it could happen that they are masked during the propagation either at the technological or at architectural level [29]. When a fault reaches the software layer of the system, it can corrupt data, instructions or the control flow. These errors may impact the correct software execution by producing erroneous results or prevent the execution of the application leading to abnormal termination or application hang. The software stack can play an important role in masking errors; at the same time, this phenomena is implicitly important for the system reliability but a hard challenge for test engineers that have to ensure safeness of their systems. As stated in the introduction, the goal of this paper is to investigate the effect of permanent faults at software layer in order to be independent from the hardware architecture running the CNN (i.e., CPU, GPU or HW accelerator). As permanent fault, we consider the Stuck-at Fault (SaF) model at 0/1 ($SaF0$ and $SaF1$). The Fault Location (FLo) is defined by (1).

$$FLo =< Layer, Connection, Bit, Polarity > \tag{1}$$

where $Layer$ corresponds to the CNN layer, $Connection$ is the edge connecting one node of the $Layer$ and Bit is one the bits of the weight associated to the $Connection.$ Finally, the $Polarity$ can be '0' or '1' depending on the SaF. The Fault Injector actually works at software layer, and its pseudo-code is provided in the $Pseudo-Code$ (1). It corresponds to a simple serial fault injector that modifies the CNN topology as described by (1).

The fault injection process consists in the following: once the CNN is fully trained, a golden run is performed collecting the golden results (aka golden_prediction), line 1 in 1. Then, the actual fault injection process is performed. The initial step requires to generate the list of faults to be injected. This fault list should be seen as a list of places where to inject the faults as described previously. Then, for any fault in the fault list (line 2 in 1), a prediction run is performed and the results collected and named as faulty_prediction. It is necessary to underline that faults are injected regardless of their polarity (stuck-at-0 or stuck-at-1). Once the fault location is fixed, the target bit is inverted (if 0 it becomes a 1 and vice-versa). In this way, we do not distinguish between the singular effect of the two fault models while obtaining a great flexibility for the huge amount of performed simulations. At this point (line 5 in 1), the obtained results are compared with the expected ones, and the results logged for a later analysis.

In details, the function *compare* of the Pseudo-code (1) classifies the prediction/classification of the faulty CNN w.r.t. the golden one. The classification is done as follows:

- **Masked:** no difference is observed from the faulty CNN and the golden one.
- **Observed:** a difference is observed from the faulty CNN and the golden one. Depending on how much the results diverge, we further classify these as:
  - **Safe:** the confidence score of the top ranked element varies by less than +/-5% w.r.t. the golden one;
  - **Unsafe:** the confidence score of the top ranked element varies by more than +/-5% w.r.t. the golden one, or the top ranked element predicted by the faulty CNN is different from that predicted by the golden one. As already discussed in [17] this is the most critical observed fault;

As reported in the Introduction Section, one of the goals of this paper is to identify the "Safe Faults Application Dependent" (SFAD) accordingly to the ISO 26262 standard. In this scenario, SFAD faults can be computed by the union between Masked and Safe-Observed fault as depicted in (2).

$$SFAD = Masked \cup Safe\_Observed\_Fault \tag{2}$$

3.1 Floating-Point Injection

In the first set of experiments, LeNet-5 connection weights are represented as single-precision binary floating-point format according to the IEEE 754 standard (Figure 4).

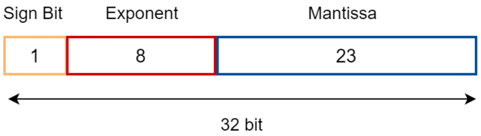


Fig. 4: IEEE 754 Single-Precision Floating-Point Standard.

The injections have been accomplished only on the four LeNet-5 layers performing arithmetic computations, i.e., the two Convolutionals and the two Fully Connected. Table 2 provides details about the configuration as well as the fault list of each layer. The first two columns of the table present the target layers; the third one specifies the amount of their connection weights. The number of possible faults is computed as the multiplication between the connections number (column 3) and the weight size (column 4) times 2 (i.e., stuck-at-0 and stuck-at-1).

Layer	Detail	Connections	Bit-Width	#Faults	#Injections
0	Convolutional	2,400	32	153,600	9,039
2	Convolutional	51,200	32	3,276,800	9,576
4	Fully Connected	3,211,264	32	205,520,896	9,604
6	Fully Connected	10,240	32	655,360	9,465

TABLE 2: LeNet-5 Fault List for the Floating-Point Injection.

As column 5 points out, the overall number is very high reflected in a non-manageable fault injection campaign execution time. Thus, in order to reduce the fault injection execution time, we randomly select a subset of faults. To obtain statistically significant results with an error margin of 1% and a confidence level of 95%, an average of 9K fault injections have to be considered. The precise numbers are given in the last column of Table 2 and they have been computed by using the approach presented in [30]. Moreover, it is necessary to underline that the injections have been performed on randomly selected bits of all the 32-bit floating-point connection weights.

3.2 Fixed-Point Injection

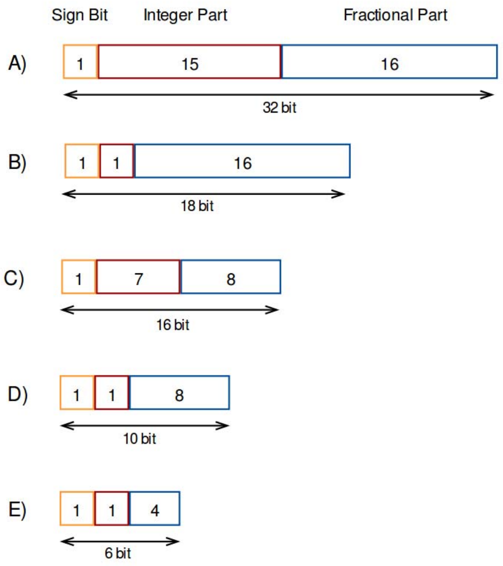


Fig. 5: Fixed-Point Data Representation.

In the second set of experiments, LeNet-5 connection weights are represented in a fixed-point format. In order to gather a wider spectrum of results, we tuned the weights bit-widths. As mentioned in Section 2.1, five different fixed-point formats of decreasing size have been chosen: from 32 up to 6 bits (Figure 5). These employ all one bit for the sign, a fixed number of bits for either the decimal part and the fractional one, i.e, those bits after the fixed-point. All the weights conversions are performed as described in Section 2.1, during the inference phase by mapping at run-time the floating-point values to fixed-point. If from one side, this technique allows to perform the experiments without resorting to external computational-intensive fixed-point libraries for the arithmetic operations; it is worth mentioning that it requires more effort during the fault injection phase. Indeed, the correct data representation should be taken into account every time a fault is placed.

The fault injections, as for the floating-point scenario, are executed only for the arithmetic layers of the network: the convolutionals and the fully connected. Table 3 details the fault list for each layer and the amount of faults that have been injected. As in the case of floating-point data format, the amount of faults is computed by multiplying the amount of the weights connections of the target layer with the bit-width of the data representation, times 2 (both the stuck-1 and stuck-0 have to be considered). These

Layer		L0	L2	L4	L6
Detail		Convolutional	Convolutional	Fully Connected	Fully Connected
Connections		2,400	51,200	3,211,264	10,240
A	Bit-width	32	32	32	32
	#Faults	153,600	3,276,800	205,520,896	655,360
	#Injection	7,680	163,840	102,760	32,768
B	Bit-width	18	18	18	18
	#Faults	86,400	1,843,200	115,605,504	368,640
	#Injection	4,080	87,040	54,591	17,408
C	Bit-width	16	16	16	16
	#Faults	76,800	1,638,400	102,771,968	327,680
	#Injection	3,840	81,920	51,380	16,384
D	Bit-width	10	10	10	10
	#Faults	48,000	1,024,000	64,225,280	204,800
	#Injection	2,160	46,080	28,901	9,216
E	Bit-width	6	6	6	6
	#Faults	28,800	614,400	38,535,168	122,880
	#Injection	1,200	25,600	16,056	5,120

TABLE 3: LeNet-5 Fault List for the Fixed-Point Injection.

details are provided for all the five experiments (A-B-C-D-E) performed by decreasing the bit-width. For sake of clarity, the injections have been performed by randomly selecting the faulty bit inside the bit-width of the weight. To lower the computational costs, only a sub-set of faults has been chosen, in accordance with the work presented in [30]. Specifically, the 5% of stuck-at faults was injected in Layer 0 - Layer 2 - Layer 6, with respect to the total amount of permanent faults. Considering the huge amount of connections of the Layer 4, it would be almost impossible to fault-simulate the 5% of faults, thus, a smaller sub-set with a reasonable amount of permanent faults was picked up.

4 Experimental Results

The main intent of this section is to report the gathered experimental results for both fixed-point and floating-point data types.

4.1 Floating-Point Representation

A first set of experiments was carried out by exploiting the *darknet* default settings, where weights are represented in a 32-bit floating-point format. All the injections have been performed considering a workload of 2023 images. Table 4 depicts the average of Unsafe Observed Fault (UOF) (the most critical ones) for each targeted layer of Lenet-5.

The experimental results demonstrate that convolutional layers are less reliable to the presence of permanent faults (Table 4). Indeed, their error percentage is about 2x times of the two fully connected layers (L4-L6). This means that permanent errors affecting the first layers of the Convolutional Neural Network negatively impact all the process of features extraction. As claimed in [17], the errors that cause large deviation into the activation values are more likely to produce a failure in output. Indeed, if faults are placed inside an initial layer, it generates a large numerical deviation, that the network is not able to correct anymore. The error is surely amplified each time the value flows through the successive layers.

A more accurate representation of the UOF percentage trend for all layers (L0-L2-L4-L6) is highlighted in Figure 6. For sake of simplicity, we plotted only charts depicting the percentage of Unsafe Observed Fault (UOF) since it seems to represent the most meaningful classification metric.

Floating-Point 32 bit	**[%] UOF**
L0 Convolutional	0.0137
L2 Convolutional	0.0139
L4 Fully Connected	0.0071
L6 Fully Connected	0.0063

TABLE 4: [%] Average of Unsafe Observed Fault (UOF) for Floating-Point Experiments.

Moreover, a deeper analysis of the results underlines that all the Unsafe Observed Faults (UOF) have been due to faults affecting the 8 bits used for storing the exponent (i.e., from bit 30 down to bit 23 of the floating-point data). The sign and the mantissa bits do not have significant impacts, i.e., they led either to Masked or Safe Observed Faults.

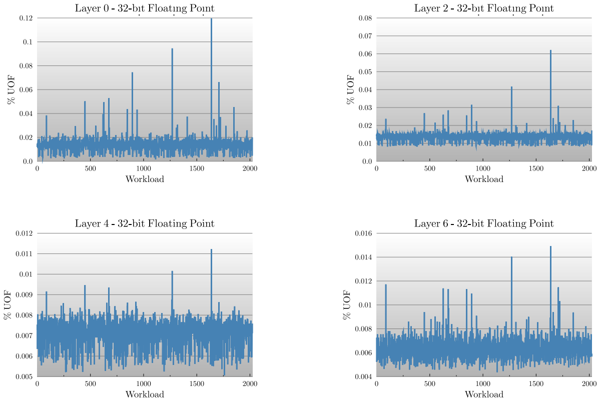


Fig. 6: [%] Percentage of Unsafe Observed Faults (UOF) for a Floating-Point CNN.

The gathered results for floating-point data are quite interesting since they are showing a different trend with respect to the effect of soft errors. Indeed, results from soft error experiments show that convolutional layers are supposed to be the more resilient to the presence of a fault, according to results shown in [17]. This is due to the fact that their role is to extract the features from the source image, while the full connected layers are supposed to be the less resilient because they classify the features extracted by the first two levels. On the other hand, these results seem to confirm the conclusion of [15] in which the authors claim this trend.

4.2 Fixed-Point Representation

Concerning the fixed-point set of experiments, all the fault injections have been performed separately on each of the above-mentioned scenarios: case A-B-C-D-E of our case study. Experimental results are provided for each of these and, most notably, led us to the following considerations:

1) Fully Connected Layers (L4 and L6) are, in principle, less critical and more resilient to permanent errors (Table 5). By comparing the Convolutional layers (L0 - L2) with the two Fully Connected (L4 - L6), it emerges that, at a first glance, for Fully Connected layers there is a reduction of more than 50% of the average error, in

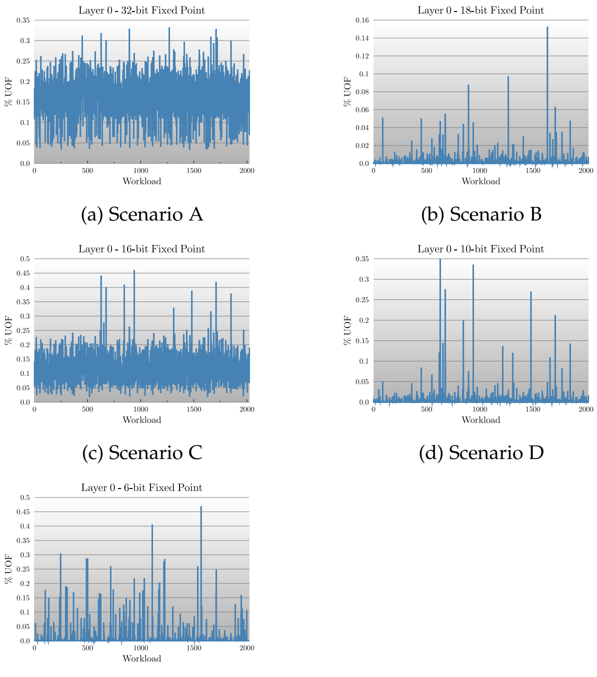
Fixed-point [%] UOF	A (32-bit)	B (18-bit)	C (16-bit)	D (10-bit)	E (6-bit)
L0 Convolutional	0.1617	0.0011	0.1126	0.0023	0.0070
L2 Convolutional	0.1513	0.0006	0.0894	0.0011	0.0034
L4 Fully Connected	0.0545	0.00004	0.0152	0.00007	0.0044
L6 Fully Connected	0.0742	0.0011	0.0530	0.0022	0.0036

TABLE 5: [%] Average of Unsafe Observed Fault (UOF) for Fixed-Point Experiments.

all the five different fixed-point representations (except for B and D, due to their peculiar structure that will be deepened in the following sections). This seems to be in line to what claimed for the floating-point results (Section 4.1).

2) The first Convolutional Layer L0 is the most critical one. Among all the four layers, it features the highest percentage of Unsafe Observed Fault (UOF). A fault affecting the first convolutional layer of a CNN, more likely produces a wrong output. In principle, it emerges that the network reliability is less impacted as the faults affect the last Fully Connected layers. In particular, Figure 7 presents a overall view of the behavior of the network when permanent faults affect L0. Indeed, for sake of conciseness, only data belonging to the first convolutional layer (L0) are depicted, for each of the considered fixed-point format (Figures 7a-7b-7c-7d-7e). What immediately comes out by observing the five graphs is the dissimilar behavior of the network when facing different bit-widths. It is immediately apparent that the worst scenarios are A and C, respectively 32-bit and 16-bit fixed-point formats, where the amount of bits reserved for the integer part is equal to those reserved for the fractional part of the weight. The average error of UOF is equal to 0.1617 and 0.1126 respectively for A and C, about 100x higher than scenarios B, C and E (see Table 5, second row).
3) Experimental results prove that there is a point where it is no longer convenient to reduce the bit-width of the weights. The last scenario E is the demonstration of the previous assertion. Indeed, in that particular case, weights are represented through only 6 bits (1 for the sign, 1 for the integer part and 4 for the fractional one). The average of UOF is from 1.6x times up to 110x times higher than B and D. This demonstrates that by lowering the bit-width, the accuracy and the reliability of the network could reduces too.

Based on these considerations, it could be claimed that *the best fixed-point bit-width choice* lies in *B* (1 bit for the sign, 1 bit for integer, 16 bits for the fractional part) and *D* (1 bit for the sign, 1 bit for integer, 8 bits for the fractional part). These two formats have in common only one bit for the integer part before the radix point. Indeed, the average percentage of UOF is the lowest between the five scenarios (see column B and D of Table 5). Particularly, the lowest percentage of UOF is obtained when adopting 18 bits for the weight representation (B). When reducing even more the bit-width up to 10 bits, the error slightly increases.



(a) Scenario A

(b) Scenario B

(c) Scenario C

(d) Scenario D

(e) Scenario E

Fig. 7: [%] Percentage of Unsafe Observed Faults (UOF) for the First Convolutional Layer (L0) with a Decreasing Bit-Width Representation.

4.3 Floating-Point and Fixed-Point Comparison

The intent of this work is to provide an overview of the reliability of a well-known CNN when a reduced bit-width is adopted for the weights. To meet the new requirements of CNNs optimization, the reliability of the target network is analyzed when two different data types are used: floating-point and fixed-point. Many conclusions can be drawn by comparing Table 4 and Table 5. According to the experimental results, a 32-bit floating point representation seems to be more reliable than a 32-bit fixed-point one (A, Table 5). The latter presents an average error that is 10 times higher that the floating-point format. All of this could be justified by the inborn structure of the two data types: the most sensible part of the floating-point turns out to be the exponential part (8 bit). On the other hand, the 32-bit fixed-point holds 15 bits for the integer part. If considering that the weights distribution groups around the zero (Section 2), 15 bits for the integer part makes the network really unreliable and error-prone. Moreover, the 32-bit floating-point representation still perform better than a 16-bit fixed-point representation (C, Table 5), where 7 bits are used to represent the integer part. Therefore, if the designer aims at reducing the bit-width of the network weight, a good idea is to adopt a fixed-point representation where only 1 bit is devoted for the integers. This outcome, as described in the Section 1.1, is consistent with the research published in [19]. Indeed, if observing the average error of B and D (Table 5), it is about 10 times lower than the 32-bit floating-point representation.

5 Conclusions

This paper presents a characterization framework for analyzing the impact of permanent faults affecting a Convolutional Neural Network intended to be deployed in automotive domain. The characterization is done by means of fault injection campaigns on the *darknet* open source framework [23]. All the experiments are performed at software level with the aim of being independent on the hardware architectures and, on the whole, to derive a common characterization of the behavior of CNNs affected by permanent faults. This could be considered an interesting outcome since the designer, starting from these results, could be able to select the most convenient data type for his application. Concerning the floating-point representation, the analyses demonstrate that it is important to take care mostly of the 8 exponent bits (i.e., from bit 30 down to bit 23) that cause critical behaviors. A redundancy technique can work well to cover only the critical parts of the system. However, the description of the test solutions as well as the redundancy techniques used for hardening the CNN is out of the scope of this work. To conclude, the proposed analysis has the advantage to be hardware independent: the designer will have the opportunity to carefully select the hardware architecture by taking into account the most critical bits on the given representation. Consequently, the test engineer will be able to focus the test efforts only on that critical parts in order to reduce the cost of the test solution.

Acknowledgments

This work has been partially founded in the context of the ODeLe IDEXLYON projet (ANR-16-IDEX-0005).

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436 EP –, 05 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[2] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[3] *Recent Advances in Deep Learning for Speech Research at Microsoft*. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), May 2013. [Online]. Available: https://www.microsoft.com/en-us/research/publication/recent-advances-in-deep-learning-for-speech-research-at-microsoft/

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484 EP –, 01 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[6] W. Sung, "Resiliency of deep neural networks under quantization." *CoRR*, vol. abs/1511.06488, 2015.

[7] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 2722–2730. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2015.312

[8] R. Cantoro, A. Firrincieli, D. Piumatti, M. Restifo, E. Sánchez, and M. S. Reorda, "About on-line functionally untestable fault identification in microprocessor cores for safety-critical applications," *2018 IEEE 19th Latin-American Test Symposium (LATS)*, pp. 1–6, 2018.

[9] M. Psarakis, D. Gizopoulos, E. Sánchez, and M. S. Reorda, "Microprocessor software-based self-testing," *IEEE Design and Test of Computers*, vol. 27, pp. 4–19, 2010.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[11] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2893356

[12] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," 2015.

[13] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016.

[14] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," 2016.

[15] F. dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, "Evaluation and mitigation of soft-errors in neural network-based object detection in three gpu architectures," pp. 169–176, 06 2017.

[16] A. Lotfi, S. Hukerikar, K. Balasubramanian, P. Racunas, N. Saxena, R. Bramley, and Y. Huang, "Resiliency of automotive object detection networks on gpu architectures," in *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1–9.

[17] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 8:1–8:12. [Online]. Available: http://doi.acm.org/10.1145/3126908.3126964

[18] Tiny-cnn. [Online]. Available: https://github.com/nyanp/tiny-cnn

[19] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[20] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64mw dnn-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, p. 1–1, 2019. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2019.2917066

[21] B. Salami, O. Unsal, and A. Cristal, "On the resilience of rtl nn accelerators: Fault characterization and mitigation," 2018.

[22] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*, 2019, pp. 1–6.

[23] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[25] [Online]. Available: https://github.com/ashitani/darknet_mnist

[26] A. Chatterjee and L. R. Varshney, "Towards optimal quantization of neural networks," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 1162–1166.

[27] R. Ding, Z. Liu, R. D. S. Blanton, and D. Marculescu, "Quantized deep neural networks for energy efficient hardware-based inference," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 1–8.

[28] [Online], "Libfixmath library," https://github.com/Petteri-Aimonen/libfixmath, 2020.

[29] M. Kooli, F. Kaddachi, G. D. Natale, and A. Bosio, "Cache- and register-aware system reliability evaluation based on data lifetime analysis," in *2016 IEEE 34th VLSI Test Symposium (VTS)*, April 2016, pp. 1–6.

[30] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 502–506.