SEU Susceptibility Analysis of a Feedforward Neural Network implemented in a SRAM-based FPGA

Israel C. Lopes, Fernanda Lima Kastensmidt and Altamiro Amadeu Susin
Instituto de Informática, UFRGS
Programa de Pós-Graduação em Microeletrônica-PGMicro
Porto Alegre, RS, Brazil.
israel.lopes@inf.ufrgs.br, fglima@inf.ufrgs.br and altamiro.susin@ufrgs.br

***Abstract*— Artificial Neural Networks (ANNs) have gained a considerable interest in clustering, pattern recognition, function approximation and many others applications, due to its parallel capability of processing the data. Moreover, ANNs can also be used to accelerate parts out of an algorithm which the data can be approximated using NPUs (Neural Processing Units). FPGAs are intrinsically parallel making these devices a good choice to implement an ANN. However, SRAM-based FPGAs are very susceptible to neutron-induced soft errors. Consequently, an ANN running in SRAM-based FPGAs devices must be characterized under soft errors. This work shows the soft error effects of a feedforward ANN under fault injection emulation performed in the bitstream of the FPGA. The number of critical bits able to provoke errors and failures in the output of the network are analyzed. Results showed the reliability superiority of the proposed ANN over the conventional implementation of the same image processing performed by the ANN. Keywords— Artificial Neural Networks, FPGA, VHDL, Image Processing.**

I. INTRODUCTION

Artificial Neural Networks (ANN) solve a plethora of difficult tasks in a parallel way, such as pattern classification, prediction, clustering, function approximation and so on [1-2]. Other recent ANN applications accelerate parts out of sequential algorithms that can be approximated [18]. The use of accelerators are becoming a promissory approach and are rapidly increasing [17]. Since the ANN advantage is processing data in parallel, hardware implementations are necessary. Although significant effort and experience is required to train ANNs, this effort can be compensated by its automatic hardware description generation due to its implementation is restricted to arithmetic operations [4][19].

This description can be implemented either in FPGA or ASIC designs. Nonetheless, an ANN datapath can be reused to perform other tasks by means the simple synaptic weights change. Therefore, due to the flexibility of FPGAs, this platform has become suited to ANN implementations. In additional, the cost and time to have a prototype of your ANN design also favor FPGA over ASIC implementations. Due to these reasons there are many ANN FPGA implementations in the literature [3-9], but they have not taken into account the soft-errors susceptibility because in old technologies these effects were slight or not perceived.

However, in recent technologies the components size reaches the energetic particle scale, thus the susceptibility is increasing as the process technology shrinks. This effect can be observed even in the ground level not only in space applications [10,11]. Soft-errors can occur in the combinational logic (Single Event Transient - SET), sequential logic and configuration bits of SRAM-based FPGA designs (Single Event Upset - SEU). Soft-errors in the configuration bits of SRAM-based FPGA has a persistent effect changing the function of some circuit until it is reconfigured [12]. Therefore, the characterization of an ANN running in a SRAM-based FPGA over soft errors permit the SEU susceptibility to be analyzed.

To achieve this purpose, this work perform a fault injection emulation in the bitstream of the proposed ANN and a conventional implementation of the same function of the ANN. The faults regions were sequentially selected by the host PC that sends the frames to the target FPGA in order the Design Under Test (DUT) to be reconfigured with the fault, then the PC receives the results and writes in the log file. Results showed a high fault mask capability for the proposed ANN in comparison with the conventional implementation. These results confirm a generic statement that ANNs are intrinsically fault tolerant, done by a related work that uses recurrent neural networks [16]. A successive image processing ANN also presented an additional fault masking capability and results showed that neurons in layers near the output are the critical parts of the network. The contribution of this paper is analyze the SEU susceptibility of feedforward neural networks that perform low-level image processing tasks. The fault tolerance of ANNs has reported in many previous publications, however these works address fault injection simulation by software or in the hardware description language (HDL) (e.g using ModelSim) [22-24]. Instead, this work perform a fault injection emulation in the bitstream, that allow errors to be provoked by hardware behavior changes. In the context of ANNs, this approach was only addressed by [16] that used another type of ANN, as it was mentioned before.

II. NEURAL NETWORK

Two mathematical morphology [13] image processing Neural Networks and its conventional implementations were

978-1-5386-0415-1/17/$31.00 ©2017 IEEE

used for the fault injection experiments. A Dilation Neural Network (DNN) was used as the Design Under Test (DUT) and an Erosion Neural Network (ENN), cascaded with the first ANN , was used to analyze the propagation of the errors. The same method was used for the conventional implementations: Conventional Dilation (CD) and Conventional Erosion (CE). The algorithm explanation of the conventional implementations are given in [13]. It is important to mention that the term "cascaded" does not mean that the DNN output is connected to the ENN input, but means that the DNN generate an image and this image is processed by the ENN. The two neural networks VHDL code were automatic generated by a program in C++ made by the author and the conventional implementations of the Dilation and Erosion processes were made by hand. The neural networks are suited to 256 colors (grayscales) bitmap images, but the basic dilation and erosion operations are performed over binary images[13][14], therefore, at this application, x00 means 'black' and xFF means 'white'. The structuring element for the dilation and erosion processes is a 3x3 square white pixels, so the inputs of the ANNs have 9 inputs of 8 bits and 1 output of 8 bits.

A. Data Representation

The precision of the weights, inputs, and activation function of neural networks implemented in FPGA has a trade-off. The more bits you use the more precise is your design, however the area utilization will increase [6-7]. Therefore, the minimum precision, that maintain the ANN accuracy, must to be found and defined. The method used to find the minimum precision consists in defining the maximum precision and gradually decreasing the signals precision until the ANN accuracy decrease. Fixed point representation were used instead of float point, because this representation demand a floating point library description and a greater area utilization [8]. As the design time, flexibility and performance were our design objectives, it was not used. The minimum activation function and weight precision found were 10 and 9 bits, respectively. As we are working with 8 bits per pixels images, the inputs has 8 bits. For the negative signals, two's complement representation was used. The inputs and the outputs bit widths of the conventional implementation is the same of the ANN implementations.

B. Activation Function Implementation

Sigmoid activation function was used in the training step. This activation function utilize a lot of area, so many alternatives were taken to deal with this issue, e.g. approximation with lines or look-up tables (LUT)[7][9]. At this application the non-linear part of the activation function were implemented with LUTs and the saturated parts were selected with comparators, as it can be seen in the Figure 1. Where "v" is the linear combiner and "y" the activation function output. The LUT address must be summed to 8514, because only positive values are accepted. When "v" is equal or lower than -8515, "y" is assigned to 0 and when "v" is equal or greater than 23660, "y" assigned to 512, otherwise the input is mapped into the LUT. Finally, the output is registered in a synchronous register.

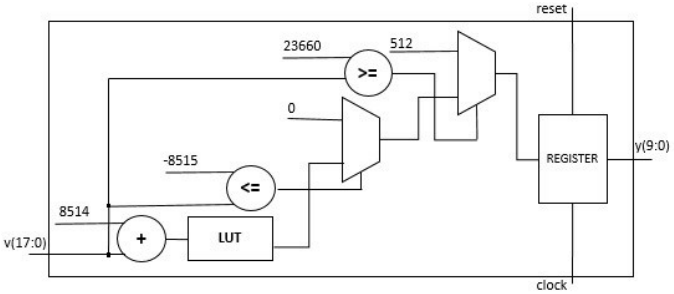

Fig. 1. Activation function component

C. Neuron Implementation

The neuron component perform the weighted sum and uses the activation function. It receives the "wi" inputs, where the "w" are the weights and "i" the index of the inputs (with 0 being the bias weight). The inputs are outputs of previous layers or external inputs for neurons of the first layer. The weights are assigned to constants at the top level. For simplicity, an example of a neuron component of 3 inputs is illustrated in the Figure 2.

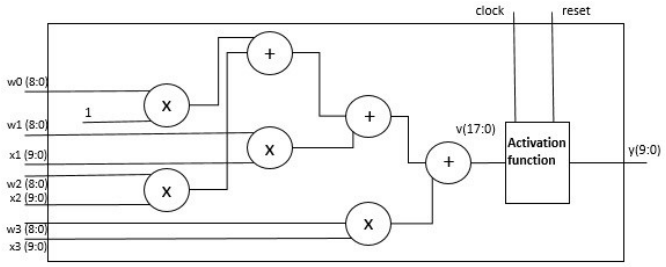

Fig. 2. Neuron component with 3 inputs

The multipliers and adders can be implemented in several ways depending on the design objective, e.g. area, power, design time, reliability and etc. The ANN design of this work aim performance, design time and flexibility. Aiming performance, the summations and multiplications are performed in parallel instead of sequential operations [4]. Aiming flexibility and design time, the adders and multipliers were described with high-level HDL code using the operators "*" and "+" of the "numeric_std" library [15], thus the utilization of LUT or DSP is defined by the synthesis strategy of the synthesis tool.

D. Network Architecture

The system top-level role is instantiate the several neuron components, connect the outputs with the inputs and assign the weight inputs to the constants defined in a package in VHDL. The benefit of neural networks implementations is that if the weight constants in the packaged is changed, other low-level image processing can be performed [4]. The top-level implementation also depends of the design objective, if area saving is the priority, only the largest layer is implemented and the inputs and weights are timing multiplexed [4-5]. Since one of our objective is performance, we chose a pipeline

implementation, where each layer take 1 cycle to compute the result.

Given that these are fully-connected ANNs [3], the number of inputs of a layer is the number of nodes of the previous layer, so one neuron per layer must be implemented to avoid area wasting by assigning inputs to 0. As the external inputs has 8 bits and the activation function has 10 bits, the bit width of the output of the first layer is lower than the others. The proposed DNN is a "9-3-1" network, where the number separated by hyphen is the number of neurons per layer, and the proposed ENN is "7-3-1". The architecture of the proposed DNN, ENN and the neuron model can be seen at Figure 3(a), 3(b) and 3(c), respectively.

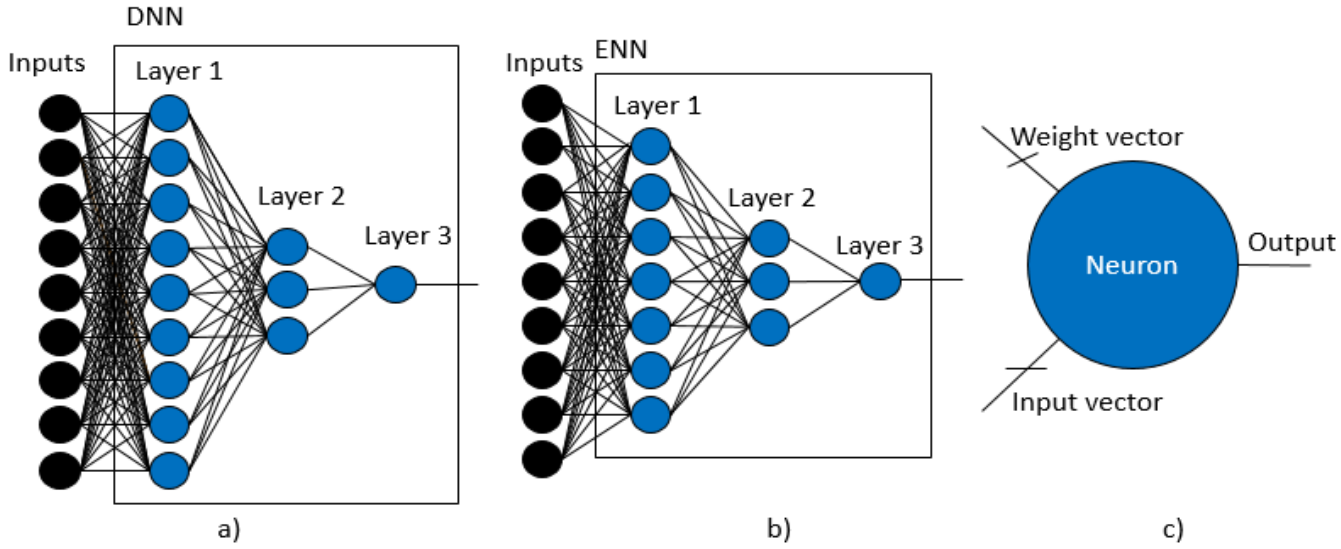

Fig. 3. a) DNN Architecture b) ENN Architecture c) Neuron model

III. Fault Injection

The Fault Injection (FI) perform the emulation of SEU effects in the configuration bits of the DUT design by partial reconfiguration. An exhaustively fault injection was performed in all the bits of the design. The faults were injected sequentially bit-by-bit in the bitstream frames. Therefore, the overall behavior works as follows. The FI system is set to a specific DUT. The FPGA is configured with the Fault Injection and DUT designs. One frame is selected and one bit of this frame is flipped. The FPGA is partially configured with this frame, the DUT is reset, execute all the dataset, and its result and fault position is registered in a log file. Then the frame is corrected and if the minor address of a frame is not the last, the minor is incremented and if the frame is not the last, the frame is incremented, otherwise the FI is ended. The FI algorithm flowchart can be seen in the Figure 4. The failure model, FI setup and FI campaigns details are described in the succeeding sections.

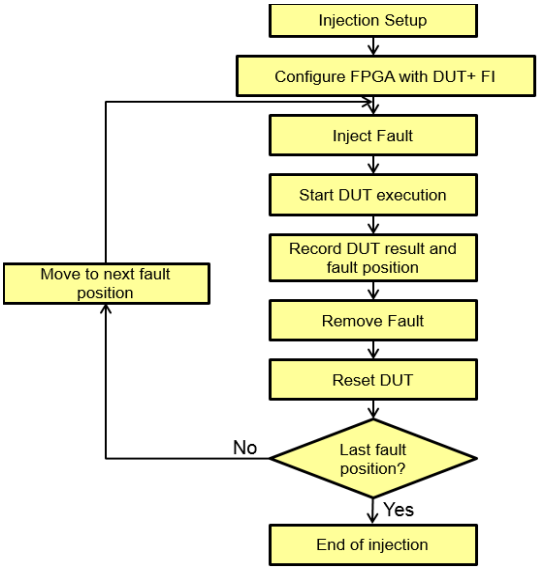

Fig. 4. FI algorithm chart

A. Failure Model

In an ANN that implement classification problems, where the major output is chosen, the failure can be simple defined as to a wrong classification and the error can be defined as to a discrepancy in the neurons outputs. However a formal definition of a failure in ANNs that perform low-level image processing is not trivial, because it depending on the application and the environment of the ANN. For example, if the application favor performance over reliability, a limited number of wrong pixels would to be accepted, but if reliability is a priority, even simple discrepancies in the pixels would to be considered a failure.

Due to this paradox, a case study was adopt to verify the propagation of the errors, as mentioned in the chapter I. Thus, an erosion ANN were implemented to verify whether the error in dilation ANN is propagated, if positive, this error is a failure. In the case study, the workload is composed by a 16x16 sub-image. Only the non-border pixels were used, executing the

proposed dilation ANN 196 times. The dilation resultant image is compared with the dilation gold to generate the errors, then the dilation resultant image is processed by the erosion ANN. The erosion resultant image is compared with the erosion gold and the failures is generated. This process is illustrated in the Figure 5. The failure model as also applied to the conventional implementation.

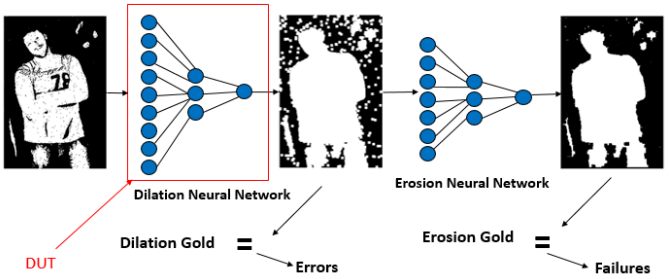

Fig. 5. Cascaded ANNs Failure model

B. *Fault injection setup*

The fault injection setup is composed by the host PC (Intel Core I5 5200 2.2 GHz 8GB RAM) that controls the fault injection and the target FPGA (Artix7 XC7A100T-1CSG324C) that receives the commands and inject the fault in the DUT configuration bits.

The FI control script was programmed in Python, it loads the bitstream file, selects the frame, flips the bit and sends the commands and the resultant frame to the ICAP by the UART communication using a USB cable. The DUT area in the FPGA is defined by the initial and final column, row, top or bottom region and minor address. The Python script programs the FPGA calling a bat file that calls the "impact.exe" software of the ISE Design Suite 14.7. The status and result of an injection is received by the serial communication and written in the log text file. The log is composed by the frame address, the bit position and the error and failure results. A simplified scheme of the Fault injection setup is shown in the Figure 6.

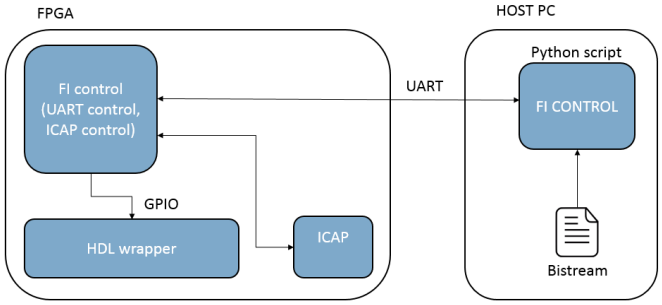

Fig. 6 Simplified Fault injection setup scheme

For the synthesis, implementation and programming of the FPGA FI Setup the Vivado 2015.4 tool was used. In the FPGA side, the FI Control has Finite State Machines (FSM) to control the ICAP communication and UART communication. The FI Control start the HDL wrapper, and wait for a "done" signal from HDL wrapper to iterate the frame bit position.

The communication between the HDL wrapper and the FI control is made by means the General Purpose Input/Output (GPIO), that is an Advanced eXtensible Interconnect (AXI) peripheral Intellectual Property (IP) Core. The MicroBlaze 9.5 soft-core processor controls the communication. A simplified HDL wrapper block design is illustrated in Figure 7, for simplicity Clock Wizard, Debug Module and Processor System Reset IP cores were omitted. The AXI Interconnect IP cores are automatic generated when an AXI peripheral is connected to MicroBlaze and Local Memory, Debug module, Clock wizard and System Processor Reset is also automatic generated when the MicroBlaze is customized. The proposed DNN and the ENN were packaged into an AXI Peripheral to be communicated with the MicroBlaze.

The MicroBlaze function is waits for a "start" signal from GPIO port, to assigns the inputs to the DNN and the ENN, compares the results with the correspondent golds, classifies the wrong outputs in either errors or failures and sends the result and the "done" signal to the FI control component by the GPIO port. For the CD FI, the same setup was used, the only change was the AXI peripheral.

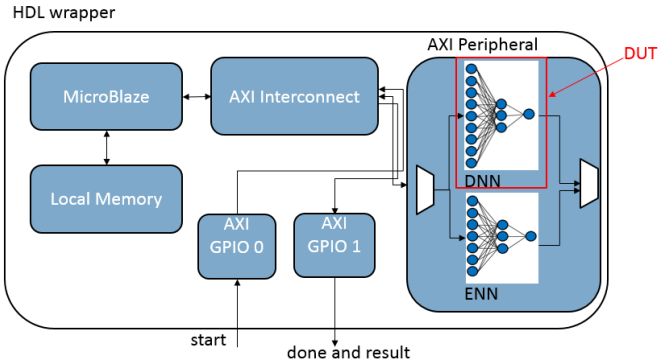

Fig. 7. Simplified HDL wrapper block design

C. *Fault Injection Campaigns*

Two fault injection campaigns were performed. In the first fault injection campaign, the area of injection was the neurons from the first layer (9 neurons) and in the second fault injection campaign, the area injected was the neurons from second and third layer (3 + 1 neurons), the correspondent area to the FI campaigns were called as to Layer 1 DNN and Layers 2-3 DNN, respectively. The area of injection were defined with pblocks and constrains were used to force the routing to be contained in the pblock to avoid fault injection on undesirable nets. The FI campaigns DUT area are shown in the Figure 8. In the conventional implementation FI only one campaign was necessary.

IV. RESULTS

A. *Performance and Area*

The frequency that meet the time constraints for the FI setup was 20MHz. Since one of the design objectives was design time, only three pipeline stages were implemented, but to implement many pipeline stages could improve the operational frequency with cost of area overhead and consequently the area

overhead could change the reliability of the ANN.

The proposed DNN and ENN area utilization are shown in the Table I and the CD and CE area utilization are shown in the Table II. The DNN area utilization was divided into 2 parts, the third row of the Table 1 presents the area utilization of Layer 1 neurons and the fourth row of Table 1 presents the area utilizations of Layers 2 and 3 neurons.

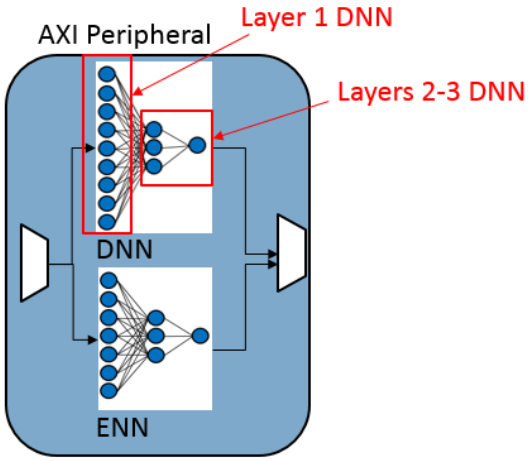

Fig. 8 FI campaigns areas

As it was explained in the section II C, in this approach, the arithmetic resources are defined by the synthesis tool strategy, since the Vivado strategy was assigned to default, the neurons did not use DSP resources, however in the area optimized strategy, DSPs were chosen. The default strategy was chosen because the target FPGA has spread DSP columns, and the implemented DUT utilizes fewer CLBs than DSPs, therefore to use a lot of DSP will underutilize CLB resources and the critical bits will not be a fair result.

TABLE I – PROPOSED DNN AND ENN AREA UTILIZATION

Design	LUT	FF
DNN	6670	127
Layer 1	4704	89
Layers 2-3	1966	38
ENN	6340	114

TABLE II - CD AND CE AREA UTILIZATION

Design	LUT	FF
CD	23	1
CE	24	1

B. Critical bits

The XC7A100T FPGA has 14663584 configuration bits, however, only a fraction of the total number of configuration memory cells are used by the design. These bits are called as to essential bits and are a subset of the configurations bits [20]. The essential bits are found out by the Xilinx Essential Bits Technology, that has an algorithm to identify the essential bits out of a configuration bits set [20]. A constraint must be defined in the Xilinx Design Constraints (XDC) file to the essential bits log be generated [21]. Thus, the fractions of essentials bits that were found to the prosed DNN and CD designs are 13.16% and 0.02%, respectively. Therefore, the proposed DNN and CD design have 1929056 and 2608 essential bits, respectively. The configuration bits that, once flipped, cause differences in the output are called as to critical bits [20]. In this paper, the critical bits are calculated over the essential bits.

The critical bits were divided into failures and errors, and these have a sub-division between single and multiple, it means one fault can corrupt many outputs from the dataset, or only one instance. Firstly, the main categories are analyzed and posteriorly a deep analyses is performed. The critical bits were computed over the essential bits available in the Xilinx report [20].

As it can be seen in the Figure 9, the total critical bits of the proposed DNN design corresponds of only 0.62% of the total essential bits, whereas the CD design corresponds of 37.96%, confirming that neural networks has a high fault masking capability [16]. The critical bits from the Layer 1 neurons correspond of 0.41% of the injected essential bits against 1.11% from the Layer 2-3 neurons, showing that the last layers are more susceptible to soft-errors than the first. It is plausible, because these are near the output.

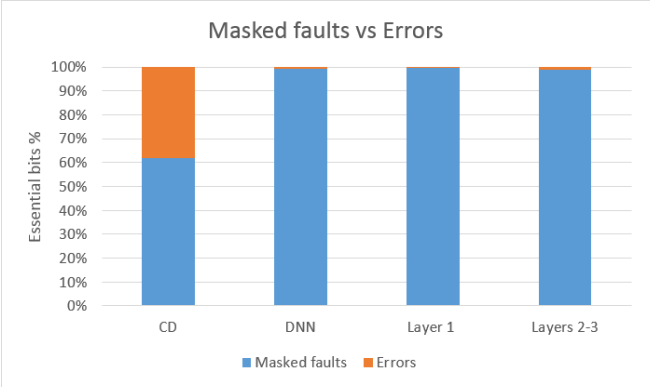

Fig. 9. Errors vs Non-Errors

Analyzing only the critical bits, the proportion of errors and failures are shown in the Figure 10. It is possible to note that in the CD more than 90% of the critical bits are failures and in the proposed DNN there is more errors than failures, therefore the cascaded network masked some errors instead of propagate them. Thus, many image processing cascaded ANNs can become an attractive fault tolerant approach. As expected, the number of failures in the Layer 2-3 neurons is greater than the Layer 1 neurons and the difference between the errors is small. Therefore, the faults provoked by the last layers are significantly enough to be propagated to the next ANN. Thus a fault tolerant approach must focus on the last layers.

A deeper analysis of the errors and failures is presented in the Figure 11, where the errors and failures are divided into single and multiple. It is clear that in the CD, almost all faults are multiple failures, whereas the proposed DNN has more multiple errors than multiple failures, showing the vulnerability of the conventional implementation.

It also possible to see that in both implementations, the single errors and failures are an insignificant portion of the errors and failures, it make sense because soft-errors in SRAM-bases FPGAs have a persistent effect, thus some configuration bit under soft-error can perform a different function to all inputs from the dataset. Therefore, scrubbing or partial reconfiguration could be good approaches to correct this kind of error.

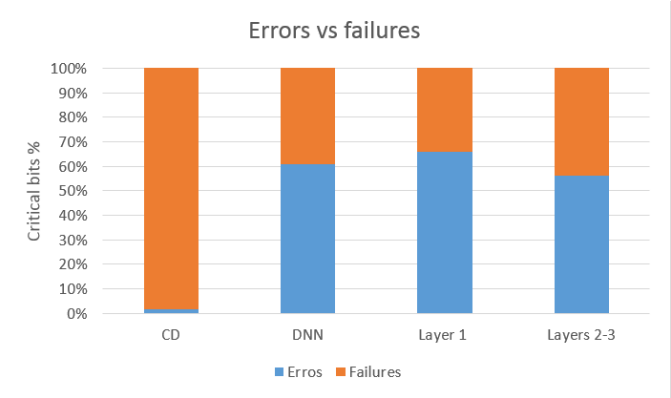

Fig. 10. Errors vs Failures

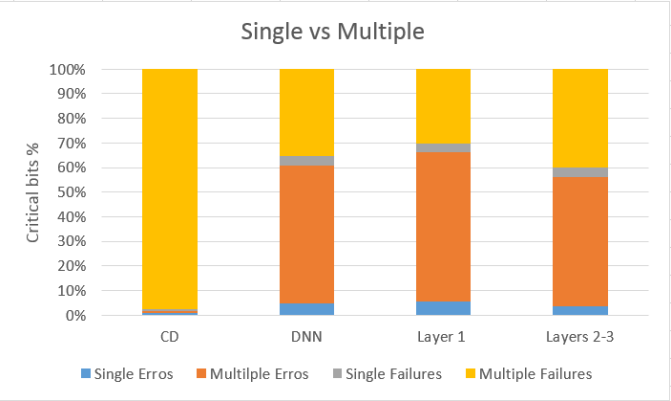

Fig. 11. Single vs Multiple Failure and Errors

V. CONCLUSION

This paper performs a bitstream fault injection in an automatic generated ANN for image processing and analyzes the critical bits showing a high fault masking capability. The cascaded Network presented an additional fault masking benefit since the image generated by the proposed DNN has no significant errors. The layers near the output were the responsible for the expressive discrepancies in the output, becoming a probably target to fault tolerant approaches, as in [16]. The proposed ANNs can be an excellent option to non-critical applications due to its intrinsically fault tolerance. However, in critical applications, image-processing feedforward ANNs could be hardened in order to achieve almost perfect fault tolerance capabilities [16].

I. ACKNOWLEDGMENT

The authors are partially supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) of Brazilian government.

REFERENCES

[1] Haykin, Simon, and Neural Network. "A comprehensive foundation." Neural Networks 2.2004 (2004).
[2] Jain, Anil K., Jianchang Mao, and K. Moidin Mohiuddin. "Artificial neural networks: A tutorial." IEEE computer 29.3 (1996): 31-44.
[3] Omondi, Amos R., and Jagath Chandana Rajapakse, eds. FPGA implementations of neural networks. Vol. 365. Dordrecht, The Netherlands: Springer, 2006.
[4] Soares, André Borin, Altamiro Amadeu Susin, and Leticia V. Guimaraes. "Automatic generation of neural networks for image processing." 2006 IEEE International Symposium on Circuits and Systems. IEEE, 2006.
[5] Himavathi, S., D. Anitha, and A. Muthuramalingam. "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization." IEEE Transactions on Neural Networks 18.3 (2007): 880-888.
[6] Zhu, Jihan, and Peter Sutton. "FPGA implementations of neural networks–a survey of a decade of progress." International Conference on Field Programmable Logic and Applications. Springer Berlin Heidelberg, 2003.
[7] Savran, Aydoğan, and Serkan Ünsal. "Hardware Implementation of a Feed forward Neural Network Using FPGAs." The third International Conference on Electrical and Electronics Engineering (ELECO 2003). 2003.
[8] Sahin, Suhap, Yasar Becerikli, and Suleyman Yazici. "Neural network implementation in hardware using FPGAs." International Conference on Neural Information Processing. Springer Berlin Heidelberg, 2006.
[9] Leekul, Prapan, Parinya Soontornwong, and Sorwat Chivapreecha. "Low complexity artificial neural network unit for sugar content detection in microwave sensor system." Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific. IEEE, 2014.
[10] Sterpone, Luca, and Massimo Violante. "A new reliability-oriented place and route algorithm for SRAM-based FPGAs." IEEE Transactions on Computers 55.6 (2006): 732-744.
[11] E. Normand, Single Event Effects in Avionics, IEEE Transactions on Nuclear 90 Science 43 (2) (1996) 461-474.
[12] Shubu Mukherjee. Architecture design for soft errors. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
[13] Gonzalez, Rafael C., and Richard E. Woods. "Digital image processing." (2008).
[14] Russ, John C. The image processing handbook. CRC press, 2016.
[15] Ashenden, Peter J. The designer's guide to VHDL. Vol. 3. Morgan Kaufmann, 2010.
[16] Clemente, Juan Antonio, et al. "Hardware implementation of a fault-tolerant Hopfield Neural Network on FPGAs." *Neurocomputing* 171 (2016): 1606-1609.
[17] Shao, Yakun Sophia, et al. "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures." *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014.
[18] Moreau, Thierry, et al. "SNNAP: Approximate computing on programmable SoCs via neural acceleration." *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015.
[19] Wang, Ying, et al. "DeepBurning: automatic generation of FPGA-based learning accelerators for the neural network family." *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016.
[20] Xilinx Inc. Soft Error Mitigation Using Prioritized Essential Bits, XAPP538 (v1.0), 2012.
[21] Xilinx Inc. Soft Error Mitigation Controller, PG036 (v4.1), 2015.
[22] Ahmadi, A., et al. "A low-cost fault-tolerant approach for hardware implementation of artificial neural networks." Computer Engineering and Technology, 2009. ICCET'09. International Conference on. Vol. 2. IEEE, 2009.
[23] Horita, Tadayoshi, Takurou Murata, and Itsuo Takanami. "A multiple-weight-and-neuron-fault tolerant digital multilayer neural network." Defect and Fault Tolerance in VLSI Systems, 2006. DFT'06. 21st IEEE International Symposium on. IEEE, 2006.
[24] Dias, Fernando Morgado, and Ana Antunes. "Fault Tolerance improvement through architecture change in Artificial Neural Networks." International Symposium on Intelligence Computation and Applications. Springer Berlin Heidelberg, 2008.
[25] Krcma, M. et al "Fault tolerant Field Programmable Neural Networks." Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC), 2015. IEEE, 2015.