Analyzing the Reliability of Alternative Convolution Implementations for Deep Learning Applications

Cristiana Bolchini, Luca Cassano, Antonio Miele, Alessandro Nazzari, Dario Passarello
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
first_name.last_name@polimi.it

***Abstract*—Convolution represents the core of Deep Learning (DL) applications, enabling the automatic extraction of features from raw input data. Several implementations of the convolution have been proposed. The impact of these different implementations on the performance of DL applications has been studied. However, no specific reliability-related analysis has been carried out. In this paper, we apply the CLASSES cross-layer reliability analysis methodology for an in-depth study aimed at: i) analyzing and characterizing the effects of Single Event Upsets occurring in Graphics Processing Units while executing the convolution operators; and ii) identifying whether a convolution implementation is more robust than others. The outcomes can then be exploited to tailor better hardening schemes for DL applications to improve reliability and reduce overhead.**

***Index Terms*—Convolutional Neural Networks, Deep Learning, Error Simulation, Fault Injection, Reliability Analysis.**

I. Introduction and Related Work

Deep Learning (DL) applications, e.g., Convolutional Neural Networks (CNNs), are increasingly being adopted for perception tasks in a large variety of application fields, also including safety- and mission-critical ones (e.g., automotive, robots, and aerospace). As an example, we may mention the Advanced Driver Assistance System in the automotive scenario [1], where lanes and tracks detection, the identification of pedestrians and obstacles, and the recognition of road signs and traffic lights are often automated via CNNs (e.g., [2]).

When designing solutions for this kind of market, though, design standards (such as the ISO 26262 for automotive [3]) specify safety-related requirements to be guaranteed. Typically, these standards request high reliability based on the system's criticality, fault detection, and management capabilities. In the mentioned application contexts, one of the sources of failures commonly considered is soft errors, e.g., Single Event Upsets (SEUs) [4]. Although such faults have traditionally been considered a concern in the space domain, it has been demonstrated that they may cause significant effects also at ground level [5], where an average rate of two SEUs every thousand billion hours has been estimated at ground level. Given an estimated number of cars traveling in Europe in 2019 at 268 million [6], we would observe a car suffering from a fault every 3.7 hours, which may be a concern. Therefore, digital systems and the executed applications exploited in this application context require reliability-related analysis and possible countermeasures.

DL applications exhibit extremely high requirements in terms of computing power due to the large number and the size of the executed elaborations and associated parameters. Therefore, classical redundancy-based fault detection and mitigation techniques, such as Duplication with Comparison or Triple Modular Redundancy, may be unfeasible because of the excessive performance overhead [7]. Recently, much effort has been devoted to developing selective-hardening and approximation-based techniques tailored for the specific DL application context to limit such costs. For example, in [8], the network layers are selectively duplicated based on the probability of a fault corrupting a layer to affect the entire application outcome.

Moreover, two peculiarities of DL applications are that i) they are inexact by nature, and ii) they generally present an internal information redundancy (in terms of layers, neurons, weights) caused by over-design w.r.t. the requirements of the specific application. From the reliability point of view, these characteristics represent novel opportunities for defining and tailoring new hardening techniques with a reduced cost and overhead. A few examples are the approach in [9], where the training process is enhanced by introducing hardware faults, and the approach in [10], where the range of corrupted values is selectively restricted to reduce the magnitude of the generated error in the final output. To be able to tailor a hardening solution that introduces a limited overhead and mitigates the actual effects of hardware faults and their criticality, an accurate fault injection/simulation tool needs to be adopted, and the mitigation strategy could focus on the most relevant and critical elements in the system.

When considering DL applications, design optimization generally targets its main operator, i.e., the *convolution*. In most CNN applications, convolution operations exploited for feature extraction from raw input data take more than 60-70% of the overall execution time. Numerous efforts have been devoted to implementing an efficient convolution operator. Among them, the three most popular ones are based on General Matrix Multiplication (GEMM) [11], on the Fast Fourier Transform (FFT) [12] and on the Winograd's filtering algorithm [13]. Given the prominent role of the operator, also reliability-related properties and characteristics should be kept into consideration for effective and possibly efficient hardening solutions. On the one hand, the performance, in terms of execution time, of the various implementations running on a

Graphics Processing Unit (GPU) have been analyzed in [14]. Furthermore, in various studies, the convolution layer has been identified as the most critical one concerning vulnerability. Past studies have partially addressed the analysis of the resilience of the some implementations, e.g., in [15], Winograd-based convolution is addressed without referring to any specific architecture. Indeed, no study analyzes in a systematic way the various implementations, as discussed in [16].

To fill this gap, this paper presents an in-depth cross-layer reliability analysis of the three previously mentioned convolution implementations with respect to hardware faults. Our goal is a comparative analysis of the resilience of different implementations of the convolution operator against SEUs affecting the GPU executing the application. To this end, a systematic and detailed analysis of the behavior of DL applications under the presence of faults is paramount, and we exploit CLASSES [17], a cross-layer methodology and framework for the analysis of the effects of SEUs affecting DL applications. The analysis is divided into two parts: i) architecture-level fault injection is exploited to understand which errors are observed on the output tensors of the CNN operators when faults are injected, and ii) application-level error simulation allows to study of how the observed errors propagate through the subsequent CNN layers to the output.

We believe that the outcomes of such analysis may enable the definition of better-tailored hardening schemes for DL applications, improving reliability with limited overhead. We first derive the most common error models for the three considered convolution implementations. Then we exploit them in error simulation on three CNN applications, namely LeNet, AlexNet, and ResNet, to understand how errors are propagated and how vulnerable the applications are to faults.

The paper is organized as follows. Section II and III present the background on CNNs and the experimental framework. Section IV and V describe the derived error models and their exploitation. Finally, last section draws the conclusions.

II. CNN and Convolutions

Convolutional Neural Network [18] are one of the most popular DL architectures. They are widely used for perception activities to extract a semantic representation from the input images and consequently to accomplish high-end tasks, such as classification, object detection, and image segmentation. CNNs are internally organized in a sequence of *layers* that process *tensors*. A tensor is a 4D structure of numerical values organized as a set of N 3D matrices; each 3D matrix is a stack of C channels, being 2D matrices with the same size, height H, and width W. Layers may include several types of operators; the most relevant is the convolution one, used to automatically "learn" and extract features from the input.

The convolution operator receives two tensors:

- $\mathbf{I}$, a set of N 3D input data matrices having sizes C_i, H_i, and W_i, and
- $\mathbf{F}$, a set of C_f 3D filters of constant weights having sizes C_i, H_f, and W_f.

The result of the convolution is a new tensor having sizes $N \times C_f \times H_o \times W_o$, where each numerical value is computed according to the following formula:

$$\mathbf{O}_{n,c,h,w} = \sum_{x=0}^{C_i-1} \sum_{y=0}^{H_f-1} \sum_{z=0}^{W_f-1} \mathbf{F}_{c,x,y,z} \cdot \mathbf{I}_{n,x,h+y,w+z} \qquad (1)$$

where $n \in [0, N)$, $c \in [0, C_f)$, $h \in [0, (H_i - H_f + 1))$, and $w \in [0, (W_i - W_f + 1))$. In this work, we focus on DL applications exploited in on-board computing, where a single image at a time is typically processed. Therefore, in the present context, $N = 1$ and thus $\mathbf{I}$ and $\mathbf{O}$ are 3D matrices. Due to the large number of values in the $\mathbf{I}$ and $\mathbf{F}$ tensors and to the complexity of the math operations in Equation 1, the convolution represents the most compute-intensive operation in the CNN model and various implementations have been investigated, among which the most relevant ones are [14]:

- **GEMM** [11]: the algorithm is based on the reshaping of the two input matrices, $\mathbf{I}$ and $\mathbf{F}$, by replicating their elements multiple times and disposing of them in such a way that the convolution operation can be computed as a General Matrix Multiplication (GEMM), whose implementation is highly optimized on GPU.
- **FFT** [12]: this algorithm exploits the Convolution Theorem stating that the Fourier transform of the convolution of two inputs is equal to the product of the Fourier transforms of the same two inputs. Thus, the algorithm applies the Fourier transform to $\mathbf{I}$ and $\mathbf{F}$ to compute the convolution as a simple multiplication.
- **Winograd** [13]: a third approach exploits Winograd's minimal filtering algorithm. The algorithm is based on specific transformations for the computation of the FIR filter to minimize the number of multiplications by replacing them with less expensive additions.

These three algorithms have been systematically analyzed from a performance point of view, targeting the GPU device. To the best of our knowledge, no similar analysis exists to evaluate their resilience against hardware faults.

III. Experimental Framework and Setup

In this work, we adopted the CLASSES methodology [17] to perform error modeling and simulation for the considered implementations of the convolution operator. CLASSES adopts a cross-layer approach that can be summarized as follows:

- **Architecture-level fault injection** is applied on the single DL operators to define errors models describing the effects of faults on the operator's output tensors, thus allowing to identify a set of recurrent corruption patterns and to define the associated error models;
- **Application-level error simulation**, exploiting the previously defined error models, is executed on the entire DL application to evaluate the propagation of the errors and the functional effects on the final outputs, thus enabling the robustness analysis of the overall application.

The strength of such an approach is to combine the advantages of both methodologies. We exploit the accuracy of fault

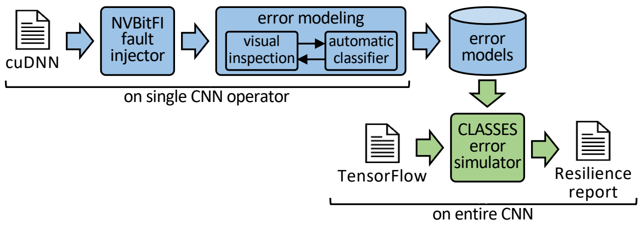


Figure 1: The experimental framework.

injection to define realistic and accurate error models and the flexibility and higher speed of error simulation in tackling the resilience analysis of an entire application.

We updated CLASSES with more recent and popular libraries and tools; the obtained framework is shown in Figure 1. Since we target NVIDIA GPUs, we adopt cuDNN [19] as DL library to implement test programs for the single convolution to be used in the architecture-level fault injection experiments. All three considered convolution algorithms are already integrated into cuDNN.

The fault injection experiments have been carried out using NVBitFI [20], a microarchitectural-level tool able to corrupt GPU registers' content during the execution of a program, i.e., the convolution operators in our case. The original version of NVBitFI offers a limited set of fault models that can capture only the effects of faults affecting the execution of a single thread. At the microarchitectural level, this fault model represents the effect of physical faults corrupting a single streaming processor, that is, the *datapath* of a single lane of the SIMD processing unit of a GPU, called streaming multiprocessor[1]. To consider also faults in the control unit, able to affect the entire group of threads executed in lock-step (called *warp*), we adopt the NVBitFI extension proposed in [22], where a warp-level fault model is defined to corrupt all the results of the 32 threads executed together. We here report the analysis results targeting a GeForce 3060 NVIDIA GPU, implementing the Ampere architecture. Finally, the definition of the error models is performed by means of the semi-automated approach proposed in CLASSES[2].

IV. Error Models Definition

For each implementation of the convolution, we executed a campaign considering 16,000 fault injection experiments for the single-thread fault model, and 16,000 for the warp-level fault model. Input tensors have been varied in every fault injection experiment to avoid any dependence of the results on the input values. Moreover, we kept the results for the two fault models separate, since there is no literature discussing the relative frequency of occurrence of the two types of faults.

For each fault injection campaign, i.e., a convolution implementation and a fault model, the corrupted output tensors have been visually inspected to extract recurrent corruption patterns. Such activity is semi-automated and applied iteratively. First, a subset of the corrupted output tensors is manually analyzed with the help of a tool that graphically shows the tensors and highlights the erroneous values. After a corruption pattern has been identified a relevant number of times, a Python classification tool is instructed to recognize it, and the entire set of corrupted tensors is then processed with such a tool. This process is repeated on the subset of not classified tensors until they all fall into a defined corruption pattern[3]. This analysis results in a set of spatial patterns that can be described algorithmically, thus representing a set of error models.

Once spatial patterns have been identified, we analyzed the domains and the magnitude of the erroneous values within the corrupted output tensors for every convolution implementation and fault model. For instance, an erroneous value may be set to Not-a-Number or a large floating point value. Each error model is annotated with this additional information.

In the following we discuss the outcomes of the two analyses, with respect to spacial patterns and value domains.

A. Spatial Patterns

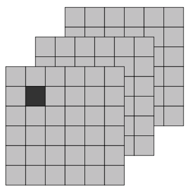
The list of recurrent spatial patterns is the first output of the previously described process for the semi-automated analysis of the corrupted output tensors; ten are the patterns emerging from the campaign, shown in Figure 2. Tables I and II report the occurrence frequencies of every corruption pattern for the three convolution implementations for the single thread and in the warp-level fault models as well as the total number of obtained corrupted tensors, respectively. The description of the ten emerged patterns is the following:

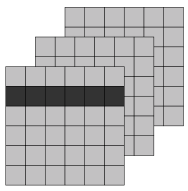
(a) **Single Point**: a single erroneous value is found in the output tensor.
(b) **Same Row**: all the erroneous values are located on the same row of a single channel. It is also possible to have non-corrupted values between two erroneous ones.
(c) **Bullet Wake**: similar to *Same Row* model, but the erroneous values are located along the channel axis. It is also possible to have channels without erroneous values on the axis.
(d) **Skip X**: erroneous values are located with a stride equal to *X* positions in the linearized tensor.
(e) **Single Block**: there is a single corrupted channel, where all the erroneous values are adjacent and the cardinality is comprised between 10 and 64.
(f) **Single Channel Alternated Blocks**: multiple nonadjacent blocks of at least 16 consecutive erroneous values located in the same channel.
(g) **Full Channel**: more than 70% of the values of the same channel are erroneous.
(h) **Multichannel Block**: an extension of the *Single Block* model where multiple channels are corrupted, each with a single block pattern.
(i) **Shattered Channel**: a combination of the *Bullet Wake* and *Single Map Random* models where erroneous values are

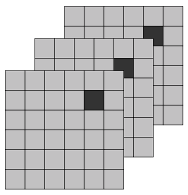
[1]For further information on the basic GPU architecture, please refer to [21] or subsequent white papers of more recent NVIDIA GPU architectures.

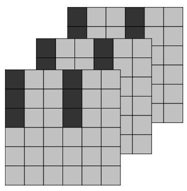
[2]The CLASSES environment, and the supporting error modeling tools can be downloaded from `https://github.com/D4De/classes`

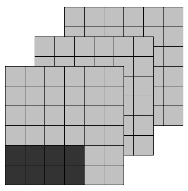
[3]The tool defined for supporting this process can be downloaded from `https://github.com/D4De/cnn-error-classifier`.

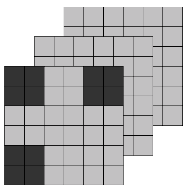

(a) Single Point

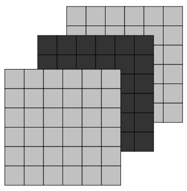

(b) Same Row

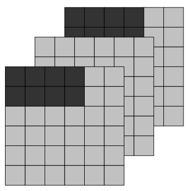

(c) Bullet wake

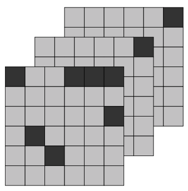

(d) Skip X


(e) Single Block


(f) Single Channel Alternated Block


(g) Full Channel


(h) Multichannel Block


(i) Shattered Channel

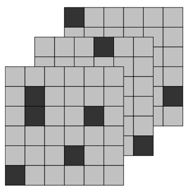

(j) Random

Figure 2: Spatial patterns for the corrupted tensors.

located on a line along the channel axis and at least one channel is randomly corrupted.

(j) **Random**: either a single channel or multiple ones are corrupted with an irregular pattern.

The two tables show that the identified spatial patterns are identical for the six fault injection campaigns, while the occurrence frequencies vary significantly.

The results reported in the tables show that the FFT implementation is the most vulnerable to faults. In almost half of the runs, the output tensor is corrupted, with a number of corrupted tensors almost twice as much w.r.t. those for the GEMM implementation, and also considerably higher than the Winograd one. Moreover, the most recurrent error pattern for the FFT implementation of the convolution is the *Full Channel*; the corruption of an entire channel will make the entire CNN produce an erroneous result with a very high probability (as discussed in detail in the next section). Indeed, a single faulty thread, or a group of 32 faulty ones (due to the warp-level fault model), affecting the computations in the frequency domain, completely corrupts the entire result in the space domain. On the other hand, when looking at the GEMM convolution, the most frequent corruption pattern is the *Single Point* when considering the single thread fault model, and the *Single Block* when considering the warp-level fault model. Indeed, GEMM implements a spatial convolution where each thread computes a single value of the output tensor; therefore, most of the time, the fault will affect a single value, or a group of 32 values, belonging to the same warp computation. In the GEMM convolution, multiple erroneous values may appear only when the fault affects shared memory resources, thus propagating errors among multiple threads. Finally, results do not show any relevant trend for the Winograd implementation. So, no conclusion can be drawn at this point.

B. Erroneous Values Domains

The second analysis we perform is on the *domains* the erroneous values belong to, to understand how faults affect the corrupted values in the output tensor and the error magnitude. This analysis can help investigate hardening solutions based on the expected values distribution, as in [23], [24]. In this perspective, it is essential to differentiate whether or not the erroneous values in the corrupted tensor fall within the nominal range. Indeed, past studies [10] showed how corrupted values sensibly exceeding the nominal range have a high probability of propagating to the final CNN output, leading to misclassification.

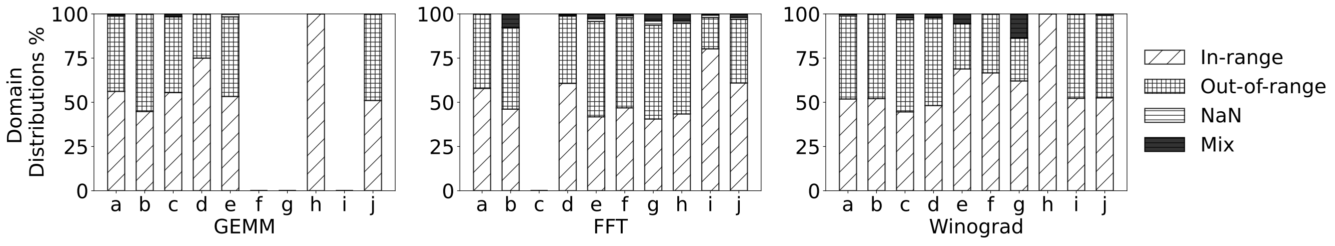
We identified three relevant erroneous values domains: i) *In-range* when the erroneous values fall within the same range

Table I: Occurrence frequencies of the corruption patterns for the single thread fault model

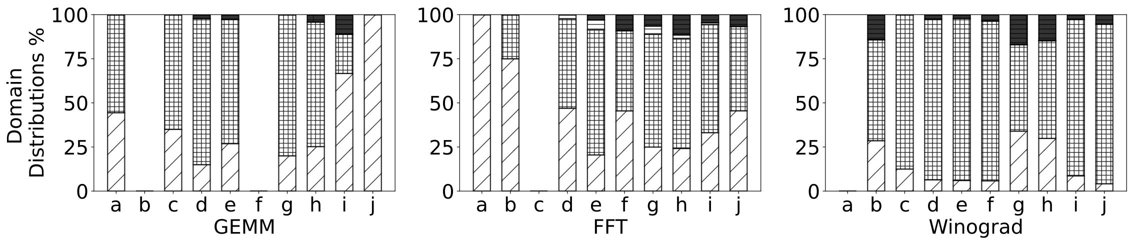
	GEMM	**FFT**	**Winograd**
a) Single Point	75.75%	0.75%	22.69%
b) Same Row	0.76%	0.34%	14.81%
c) Bullet Wake	20.57%	0%	5.96%
d) Skip X	0.1%	3.13%	17.75%
e) Single Block	1.56%	17.65%	2.14%
f) Single Channel Alternated Blocks	0%	1.27%	0.06%
g) Full Channel	0%	68.19%	2.89%
h) Multichannel Block	0.03%	1.33%	0.09%
i) Shattered Channel	0%	2.04%	15.11%
j) Random	1.23%	5.30%	18.50%
# corrupted tensors	3835	7613	4665

Table II: Occurrence frequencies of the corruption patterns for the warp-level fault model

	GEMM	**FFT**	**Winograd**
a) Single Point	0.37%	0.01%	0%
b) Same Row	0%	0.05%	0.12%
c) Bullet Wake	1.25%	0%	0.14%
d) Skip X	20.82%	1.54%	15.34%
e) Single Block	63.31%	11.3%	5.12%
f) Single Channel Alternated Blocks	0%	0.38%	4.33%
g) Full Channel	0.1%	70.5%	9.21%
h) Multichannel Block	13.93%	8.9%	9.96%
i) Shattered Channel	0.19%	5.76%	59.06%
j) Random	0.03%	1.56%	5.72%
# corrupted tensors	4803	8784	5870



(1) Single thread faults



(2) Warp-level faults

Figure 3: Distribution of erroneous values domains in the corrupted tensors.

of the correct values; ii) *Out-of-range*, when we observe a positive/negative *spike*, with the erroneous values exceeding the expected range; and *Not-a-number (NaN)*. Moreover, in several corrupted tensors, we observed a combination of erroneous values belonging to two or more of the above classes; we label this case as *Mix*.

Figure 3 reports the distributions of the erroneous values domains for every convolution implementation and every spatial distribution for the single thread and the warp-level fault models. For the sake of space, in the plots, we used the same alphabetic labels for the spatial distribution as in Figure 2. When considering the single thread fault model, it can be noticed that the erroneous values are almost equally distributed between *In-range* and *Out-of-range*, with an always limited percentage of *Mix* cases. On the other hand, when considering the warp-level fault model, we can observe a more significant percentage of *Mix* cases. However, most erroneous values fall in the *Out-of-range* domain. Finally, *NaN* corruptions appear in about 0.6%, 4%, and 0.8% of the cases for the GEMM, FFT, and Winograd convolutions, respectively.

V. Application-level Analysis and Exploitation

We exploited the defined error models in application-level error simulations of three CNNs, namely, LeNet, AlexNet, and ResNet, to study how the resilience of the overall application varies when considering different convolution implementations. For each CNN and each fault model, i.e., single thread and warp-level, we ran an error simulation campaign consisting of 10,000 experiments. In each campaign, the error models have been *injected*, assuming the occurrence probabilities for the spatial distributions and the erroneous values domains previously presented. Results are reported in Table III providing the operator vulnerability, i.e., the probability that a hardware fault produces an error on the operator output and the percentage of misclassifications over the total number of error simulations. Operator vulnerability has been computed through the results of architecture-level fault injection reported in Table I and II; indeed, these values are independent of the specific CNN application.

Table III: Error simulation results

Impl.	**Fault Model**	**Op. Vuln.**	**% Misclassifications**		
			LeNet	AlexNet	ResNet
GEMM	thread	24%	20.09%	36.13%	27.85%
	warp	30%	81.63%	83.21%	75.99%
FFT	thread	48%	81.84%	88.09%	59.02%
	warp	55%	85.42%	94.38%	77.20%
Winograd	thread	29%	43.90%	49.38%	39.55%
	warp	37%	90.87%	95.52%	89.59%

As a first consideration, the FFT convolution presents the highest operator vulnerability and the highest percentage of misclassifications when considering single-thread faults. When considering the warp-level fault model, we may notice a very high misclassification probability for all the convolution implementations, thus confirming that the corruption of 32 threads has a more disruptive effect. More in detail, the FFT convolution is the worst for LeNet and AlexNet, while for ResNet, the Winograd convolution presents results worse than the other two implementations.

Using the adopted two-level analysis framework, we better investigated the actual causes of these results. First, we notice that the erroneous values domain highly affects the capability of the CNN to classify correctly. For every convolution implementation, Table IV reports the misclassification probability associated with the four erroneous values domains and irrespective of the spatial patterns (for the sake of space, these values have been averaged on the three CNN applications). It

Table IV: Mis-classifications vs. Erroneous Values Domains

Domain	GEMM		FFT		Winograd	
	thread	warp	thread	warp	thread	warp
In-Range	5.22%	27.82%	50.29%	55.72%	14.30%	40.57%
Out-of-Range	62.28%	85.12%	93.15%	93.23%	74.92%	87.59%
NaN	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Mix	38.73%	53.97%	73.49%	73.57%	59.06%	53.97%

Table V: Mis-classifications vs. Spatial Patterns with Clipping

Spatial pattern	GEMM		FFT		Winograd	
	thread	warp	thread	warp	thread	warp
Single Point	2.00%	2.11%	2.61%	0.00%	2.02%	-
Same Row	8.76%	-	10.83%	0.00%	3.46%	6.56%
Bullet Wake	13.70%	7.82%	-	-	11.62%	8.13%
Skip X	14.53%	14.65%	28.28%	43.26%	15.35%	35.88%
Single Block	34.59%	26.40%	32.43%	32.29%	30.10%	27.49%
Single Channel Alt.	-	-	36.40%	34.78%	14.07%	35.95%
Full Channel	-	27.60%	57.90%	60.78%	57.42%	63.68%
Multichannel Block	22.97%	56.26%	53.05%	49.84%	59.51%	56.11%
Shatter Channel	-	44.68%	54.89%	57.43%	28.77%	38.23%
Random	21.79%	19.11%	35.27%	43.48%	16.45%	36.31%

can be noticed that *NaN* values always cause misclassifications; at the same time, *Out-of-range* erroneous values cause misclassification in up to the 93.15% of the cases, while the effect of *In-range* errors is much less severe.

To handle misclassifications, past approaches, e.g., [10], proposed clipping-based strategies to clamp the erroneous values within the nominal range. Nevertheless, these approaches have not been validated on accurate error models; single erroneous values are generally only considered. As we demonstrated in the previous section, more complex spatial patterns, counting on multiple erroneous values, may be produced by faults. Therefore, we analyzed how effective clipping-based error correction methodologies would be when dealing with errors counting multiple erroneous values. For the three convolution implementations, Table V reports misclassification probabilities associated with every spatial distribution after applying a clipping strategy (again, these values have been averaged on the three CNN applications). It can be observed that, as reported in the literature, the effect of single erroneous values is absorbed in most cases; on the other hand, more complex spatial patterns still cause misclassifications regardless of the clipping.

VI. Conclusions

We presented an in-depth analysis of the effects of faults on three convolution implementations accelerated on GPUs. First, we extracted a rich and realistic set of error models defined in terms of spatial distribution and domains of the erroneous values and the associated occurrence probabilities. Then, we identified GEMM as the most resilient convolution implementation. Finally, we demonstrated that error correction approaches based on clipping are ineffective against complex errors presenting multiple erroneous values and that more advanced strategies are still required.

References

[1] M. Campbell, M. Egerstedt, J. How, and R. Murray, "Autonomous driving in urban environments: approaches, lessons and challenges," *Philosophical Trans. of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4649–4672, 2010.

[2] Z. Ouyang, J. Niu, Y. Liu, and M. Guizani, "Deep CNN-Based Real-Time Traffic Light Detector for Self-Driving Vehicles," *IEEE Trans. Mobile Computing*, vol. 19, no. 2, pp. 300–313, 2020.

[3] Int. Organization for Standardization (ISO), "26262: Road vehicles - Functional safety," https://www.iso.org/standard/68383.html, 2011.

[4] T. Karnik and P. Hazucha, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, 2004.

[5] E. Normand, "Single event upset at ground level," *IEEE Trans. Nuclear Science*, vol. 43, no. 6, pp. 2742–2750, 1996.

[6] ACEA - European Automobile Manufacturers' Association, "ACEA Report: Vehicles in use - Europe 2019," https://www.acea.be/publications/article/report-vehicles-in-use-europe-2019, 2019, (Accessed on 03/20/2020).

[7] F. F. dos Santos, L. Carro, and P. Rech, "Kernel and layer vulnerability factor to evaluate object detection reliability in GPUs," *IET Computers & Digital Techniques*, vol. 13, no. 3, pp. 178–186, 2019.

[8] A. Mahmoud, S. K. S. Hari, C. W. Fletcher, S. V. Adve, C. Sakr, N. Shanbhag, P. Molchanov, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing Selective Protection for CNN Resilience," in *Proc. Int. Symp. Software Reliability Engineering*, 2021, pp. 127–138.

[9] N. Cavagnero, F. F. dos Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, "Transient-Fault-Aware Design and Training to Enhance DNNs Reliability with Zero-Overhead," in *Proc. Symp. On-Line Testing and Robust System Design*, 2022, pp. 1–7.

[10] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for Deep Neural Networks through range restriction," in *Proc. Int. Conf. Dependable Systems and Networks*, 2021, pp. 1–13.

[11] S. Chetlur, C. Woolley, P. Vandersmersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[12] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, Yann LeCun, "Fast Convolutional Nets With fbfft: A GPU Performance Evaluation," in *Proc. Int. Conf. on Learning Representations*, 2015, pp. 1–17.

[13] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4013–4021.

[14] M. Jorda, P. Valero-Lara, and A. J. Pena, "Performance Evaluation of cuDNN Convolution Algorithms on NVIDIA Volta GPUs," *IEEE Access*, vol. 7, pp. 70 461–70 473, 2019.

[15] X. Xue, H. Huang, C. Liu, T. Luo, L. Zhang, and Y. Wang, "Winograd convolution: A perspective from fault tolerance," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[16] C. Bolchini, L. Cassano, A. Miele, and A. Nazzari, "Selective Hardening of CNNs based on Layer Vulnerability Estimation," in *Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2022, pp. 1–6.

[17] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and Accurate Error Simulation for CNNs Against Soft Errors," *IEEE Trans. on Computers*, vol. early access, pp. 1–14, 2022.

[18] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[19] NVIDIA, "cuDNN," https://developer.nvidia.com/cudnn, accessed: 2023-05-05.

[20] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs," in *Proc. Int. Symp. Microarchitecture*, 2019, p. 372–383.

[21] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[22] F. F. dos Santos, J. E. Rodriguez Condia, L. Carro, M. Sonza Reorda, and P. Rech, "Revealing GPUs Vulnerabilities by Combining Register-Transfer and Software-Level Fault Injection," in *Proc. Int. Conf. Dependable Systems and Networks*, 2021, pp. 292–304.

[23] E. Ozen and A. Orailoglu, "Sanity-check: Boosting the reliability of safety-critical deep neural network applications," in *Asian Test Symp.*, 2019, pp. 7–75.

[24] C. Amarnath, M. Mejri, K. Ma, and A. Chatterjee, "Soft Error Resilient Deep Learning Systems Using Neuron Gradient Statistics," in *Proc. Intl. Symp. On-Line Testing and Robust System Design*, 2022, pp. 1–7.