Are CNNs reliable enough for critical applications? An exploratory study

Mohamed Ayoub Neggaz, Ihsen Alouani, Smail Niar, Fadi Kurdahi

▶ **To cite this version:**

Mohamed Ayoub Neggaz, Ihsen Alouani, Smail Niar, Fadi Kurdahi. Are CNNs reliable enough for critical applications? An exploratory study. IEEE Design & Test, 2020, 37 (2), pp.76-83. 10.1109/MDAT.2019.2952336 . hal-03321520

HAL Id: hal-03321520
https://hal.science/hal-03321520v1

Submitted on 21 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IEEE Design & Test 37-2 (2020) 76-83

Are CNNs Reliable Enough for Critical Applications? An Exploratory Study

Mohamed A. Neggaz, Ihsen Alouani, and Smail Niar
Université Polytechnique Hauts-De-France

Fadi Kurdahi
University of California at Irvine

DEEP LEARNING SYSTEMS such as convolutional neural networks (CNNs) have shown remarkable efficiency in dealing with a variety of complex real-life problems. These techniques have been found to be deployed in widespread domains from main-stream applications to safety-critical systems. From handwritten digit recognition to advanced environ-mental perception for autonomous cars, deep neu-ral networks (DNNs) have demonstrated an effective ability to train robust feature extractors that can be successfully exploited by a classifier.

In the context of performance-driven design requirements, new hardware generations continuously shrink the transistor dimensions, thereby increasing circuits' sensitivity to external events which can negatively affect their reliability. One of the major sources of these errors in modern embedded systems are soft errors such as single-event upsets (SEUs) and single-event transients (SETs) that are typically caused by high energy particles striking elec-tronic devices. These events can lead to bitflips in sequential parts and mem-ory cells. This situation may propagate to cause system-level fail ures and viola-tion of safety specifications. In safety-crit-ical systems, incorrect values represent a serious concern, as these systems must comply with strict safety standards.

Intentional attacks are another potential source of faults. The widespread usage of CNNs led to the development of sophisticated attacks. Adversarial attacks are among these attacks. Malicious users could intentionally tamper with processed data to fool the network. While these attacks are limited to the input, they can be easily generalized to other parameters of the system, such as the CNN weights [1].

Given the trend of high performance, sensitive hardware platforms and reliability issues of CNN-hosting systems remain an underexplored topic yet. In fact, since CNNs can be dedicated to safety-critical applications, one cannot rely on their inherent fault tolerance aspect without deep exploration. The reliability of CNNs, especially those dedicated to safety-critical applications, should be a concern in the early design stage, not an afterthought.

In this work, we consider random errors resulting from the environment. These errors are simulated as bit-flips. Redundancy is a common solution to

DOI : 10.1109/MDAT.2019.2952336

reliability issues. However, a systematic redundancy has high resource and energy overheads, and is not suitable for limited-budget systems [2].

We undertook an extensive experimental study, involving scenarios with different levels of error injection. We showed the following.

- Our experimental results successfully characterize the distribution of errors in layer-wise parameters of CNNs.
- Our approach shows that the quantization has, counterintuitively, a positive impact on CNNs' resilience to errors.
- This article explores the impact of weights' bit significance on the error resilience of CNNs. We show that one single bit, namely the most significant bit of the exponent, needs hardening in floating-point-based CNNs. Other bits are insignificant from a reliability impact perspective.
- Our approach can be used to construct a set of reliability guidelines for the deployment of CNNs in critical and aggressive environments.

This article also presents a fault injection engine operating on CNN weights. This engine studies different reliability issues of a given trained CNN. The source is made publicly available.[1]

Related works

Two types of fault injections were presented by Liu et al. [3]. They managed to achieve misclassification after a series of careful bit-flipping. They reported the loss of accuracy for the target class only. In our work, we study the impact on the overall accuracy. Furthermore, they assumed that the injections are selected carefully, whereas in our experimental setup, injections are performed randomly to simulate environment faults.

In [2], a method for estimating fault tolerance in artificial neural networks (ANNs) is proposed. This method exploits the redundancy of hidden units to increase the network's fault tolerance. In their results, a very high number of replications (more than 7) are needed to achieve complete fault tolerance. Our study locates the most vulnerable parts to reduce this overhead when redundancy techniques are employed.

The partial fault tolerance (PFT) of ANNs during the training was discussed in [4]. The authors considered replication to enhance the PFT of a network. In [5], it was shown that only 17 bit-flips are required to corrupt a network such as Alexnet. The authors carefully selected the target bits to be flipped. In this work, we focus on random error injections at different levels: data representation, position in the representation, and position in the architecture.

The inherent fault tolerance of networks has also been studied in [6]. However, the authors focused on relatively small CNNs. Their methodology is based on stuck-at faults. Stuck-at faults in feedforward neural nets were also discussed in [7]. Replication was proposed as a solution to achieve fault tolerance.

Stuck-at faults were also discussed in [8]. The authors studied the impact of faulty multiply accumulate (MAC) units on the tensor processing unit (TPU)'s grid-like architecture. Their results show that with less than 0.006% fault rate, the accuracy degrades dramatically. They also proposed two solutions by pruning and retraining. Their study considered only permanent errors in activation, since they claim that memory errors could be mitigated by error correcting codes (ECCs).

The reliability of object detection networks on GPUs has been studied in [9]. This study was based on fault injection and exploited the error-leaking potential between GPU threads. Our study is platform-independent, and the results could be projected for other embedded systems.

To the best of our knowledge, this is the first study that explores random fault injections in CNNs' weight memory, considering different quantization parameters, different data representations, the bit position, and the layer of occurring faults.

Experimental methodology

In this section, we present our setup and methodology to evaluate the reliability. We use the same methodology as in [10] with a different experimental setup. When compared to the previous article, we evaluate more variables and confirm the obtained results on other networks.

Methodology

Without considering the physical damage, soft errors compromise system functionality by causing bit-flips in memory or in computational elements. Since memory errors are more critical and durable [10], we only focus on bit-flips in memory. In most machine learning accelerator designs, two memories are present: 1) the weights' memory ($\mathcal{M}_w$), which stores trained network parameters and 2) the

[1]https://github.com/cypox/CNN-Fault-Injector

intermediate output memory ($\mathcal{M}_i$), which stores the output of hidden layers.

$\mathcal{M}_i$ receives new values for each input. A bit-flip in this memory will only affect the current run, and only if it occurs before the subsequent layer starts processing. This is similar to the errors in computational parts, which is not discussed in this article. On the other hand, a bit-flip in $\mathcal{M}_w$ will remain active until a new network is deployed. We focus on this kind of errors.

To reproduce this behavior, we simulate a soft error in $\mathcal{M}_w$ by a number of bit-flips in a random weight, once the network is trained. Multiple studies are conducted based on this simulation hypothesis. In each study, we evaluate the robustness variable of a CNN-based system. The position of the flip is decided by the study and the evaluated variable.

- *Networks:* CNNs can perform a variety of tasks. Whether it is for images, voice, text, or other input types, classification is the most common task performed by CNNs. Other tasks, such as detection, uses a classification subnetwork. In the experimental setup we propose, we only consider classification networks. Consequently, the study can be projected for other variants.
- *Data set:* Measuring a CNN's accuracy requires a labeled test set. Since test sets are usually not labeled, we use the validation set of ImageNet, as used in the challenge. The set contains 50,000 images with the corresponding class of each image. The set contained 1000 classes.
- *Data representation:* We consider two data representations:
  - *IEEE-754's 32-bit float:* This is the standard representation format for the floating-point format. It is the dominant representation in CPU and GPU architectures. Many GPUs are optimized to deal with floating-point multiplications. For simplicity, we refer to this representation as $\mathcal{F}$ in the rest of this article.
  - *X-bit fixed point:* We used the format from [11]. Trading accuracy for high performance by using low bit-widths is a common practice in CNN acceleration. This representation uses two parameters: bit-width and fractional length. Negative fractional lengths can be used to represent powers of two. This representation is referred to as $\mathcal{Q}$ (for quantized) in the rest of this article.
- *Injection algorithm:* Based on the fault-injection model in [10], we create a fault-injection engine. The engine takes a trained network, a data set, and a test type. The test type dictates the execution flow and the parameters to vary during the test. Multiple test types are developed, and more details are provided later in this section. Depending on the selected test type, a series of bit-flips are performed in the network's weights. After each test, the engine reports the measured accuracy on the data set after the injection.

We consider three test types: *full-network*, *index-wise,* and *layer-wise* tests.

- *Full-network injection:* The engine generates a list of errors that are identified by their layer and their position in the layer. The engine incrementally injects errors in the network. After each injection, we measure the accuracy of the whole data set. As a result, we aim to compare the two data representations in terms of inherent resilience. This comparison is useful to decide which data representation is more suitable when faults are present.
- *Indexed injection:* In this test, the generated errors are injected in a fixed bit significance. The engine then loops over every possible position from the least to the most significant bit.[2] The result of this test type extends to the result of the *full* test. After comparing the two representations, we use this study to explain the difference, if any. Furthermore, this helps localizing the most vulnerable bits to protect.
- *Layer-wise injection:* In this test, errors are generated in the same layer with different positions. This test is repeated for each layer while reporting the accuracy after each run. The number of errors injected is proportional to the number of parameters of each layer. This is similar to the real world, where the soft-error rate is proportional to the surface of the chip. This study allows us to understand the inherent tolerance of CNN layers. Finding the most vulnerable layers will assist in creating comprehensive reliability enhancement strategies.

These tests are repeated 60 times. In each run, the engine generates a new set of errors and the injection of the generated errors is performed each run.

[2]In the case of a trained network represented as 32-bit floating point, the engine loops over the 32-bit positions.

We then present the mean of 60 runs as well as the maximum, the minimum, and the standard deviation of the test.

A single soft error can cause multiple bit-flips. Furthermore, memory errors are cumulative. We fix the number of errors to be injected when varying the index of the bit-flip to 50. For layers, we inject errors proportional to the number of parameters with at least one injected error.[3] An extensive study, with a variable number of errors, is possible; however, the same tendency reappears.

Experimental setup

The engine was developed in Python. For CNN inference, we used the framework Caffe. The code is made publicly available.[4] As part of our study, we performed injections on quantized (low-precision) CNNs. The weights were obtained using Ristretto.

The experiments were performed on an Nvidia Quadro P5000 GPU with an Intel Xeon W-2123 CPU with a 3.60-GHz frequency.

[3]Scales with the size of the target layer.
[4]https://github.com/cypox/CNN-Fault-Injector

Convolutional neural networks

We used four network architectures: GoogleNet, Alexnet, VGG16, and SqueezeNet. These networks were selected for their wide usage, diversity, various sizes, and high accuracy. Their convolutional layers are widely reproduced as a feature extractor in other models. This facilitates generalizing the obtained results to other networks.

For the 32-bit floating point-represented weights, we used trained instances from Caffe's Model Zoo.[5] We used Ristretto to quantize and fine-tune the four networks into 8-bit fixed point networks without a huge loss in accuracy.

Experimental results

The results we collected from the engine are presented in this section. For each test type (*full*, *layer*, and *index*), we show the obtained results separately.

Impact of data representation and quantization

The results were obtained on weights represented as 32-bit floating points. We present a comparison of

[5]Publicly available on: https://github.com/BVLC/caffe/wiki/Model-Zoo

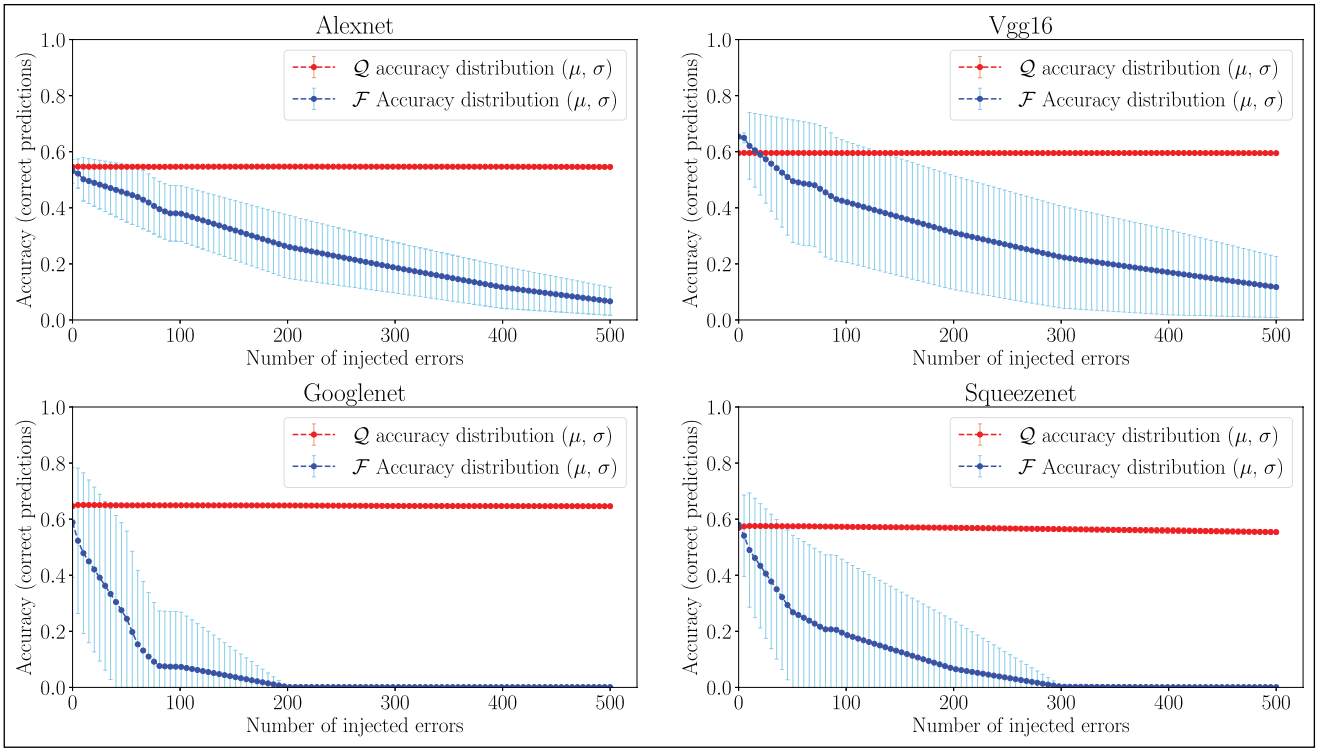

Figure 1. Comparison between the 8-bit fixed-point representation ($\mathcal{Q}$) of weights and the 32-bit IEEE-754 representation ($\mathcal{F}$). The results of different runs are presented as the mean and the standard deviation of the top-1 accuracy.

Table 1. Number of weights per network.

Network	Alexnet	VGG16	Googlenet	Squeezenet
Weights ($\times 10^6$)	60.97	138.36	7	1.25

the impact of different data representations on the accuracy of different networks.

Figure 1 illustrates the result of comparing the two representations. The $\mathcal{Q}$-representation is clearly more resilient than its counterpart. This tendency is present for the four networks with different rates. The theoretical reason behind this resilience is explained by the overall difference after injection, denoted $\mathcal{A}$ in [10]. For instance, the $\mathcal{Q}$-representation with seven decimal bits and one integer bit will differ from the original value by at most ±1. For the $\mathcal{F}$-representation, the difference in activations can reach 3×10^{38} [10].

The decrease in the accuracy in VGG16 and Alexnet is not as fast as the decrease for the same number of errors in GoogleNet and squeezeNet. The main reason for this phenomenon is the number of weights, as shown in Table 1. The same number of errors has less impact if the number of weights is important.

Significance of bits

To further explore this decrease in accuracy, we investigate the individual impact of the bit position. The injections are performed at the same position on the four networks for each run. The only difference is the index of the bit-flip on the binary representation of the weight. We performed this study only on the $\mathcal{F}$-representation. The $\mathcal{Q}$-representation in invulnerable to bit-flips, as shown in the previous results in Figure 1.

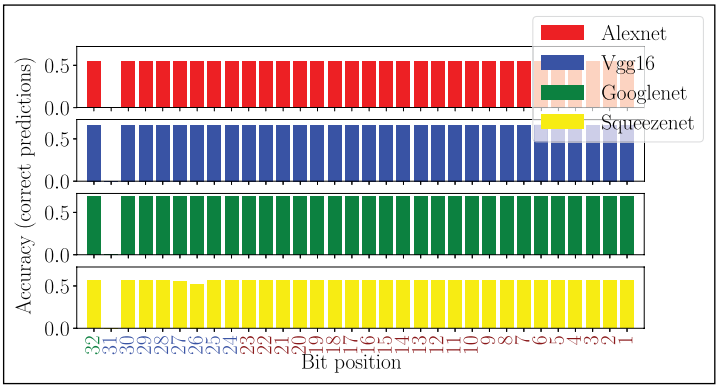

Figure 2. Position of bit-flips in the $\mathcal{F}$-representation and its impact on the accuracy. In the *X*-axis, red labels represent the mantissa, blue labels represent the exponent, and the sign bit is in green.

In Figure 2, the four networks show the same tendency. Unless bits are injected in the exponent's most significant bit, almost no impact on the accuracy is perceived.

Layer tolerance

The impact of injected faults may depend on its location within the network architecture. This section explores the layers' tolerance aspect. Similar to that discussed in the "Significance of bits" section, we isolate the target layer in the fault injection process. This isolation allows tracking the individual impact of the chosen layer on the overall accuracy.

GoogleNet and SqueezeNet have a special architecture. They are built on top of two modules, *inception* for the former and *fire* modules for the latter. These modules regroup a set of convolutional layers working in parallel on the same input. The output of the module is obtained by concatenating the outputs of each execution branch. For clarity, we reduced the individual layers into the corresponding modules. For each module, we take the average accuracy of its individual layers.

Figure 3 presents the results of this study. The four networks tend to lose more accuracy when injections occur in advanced layers. This is correlated to our previous results in [10]. While CNNs have a sequential structure, error propagation is not problematic in CNNs. Errors in early layers have, in general, less impact on the accuracy. This shows the implicit characteristic of CNNs to maintain a same behavior when incorrect values are forwarded. This is explained by the implicit redundancy in CNN weights. After training, many weight clusters are repeated. A small number of errors are found if they occur in the first layers. Techniques such as pruning can greatly affect this study. They explore weight redundancy to reduce computations. While they achieve high throughput with acceptable accuracy, reliability can be greatly compromised [12]. This tradeoff should be considered to evaluate the CNN acceleration in aggressive environments.

It is worth mentioning that, although the mean value is at a comfortable accuracy, the minimum

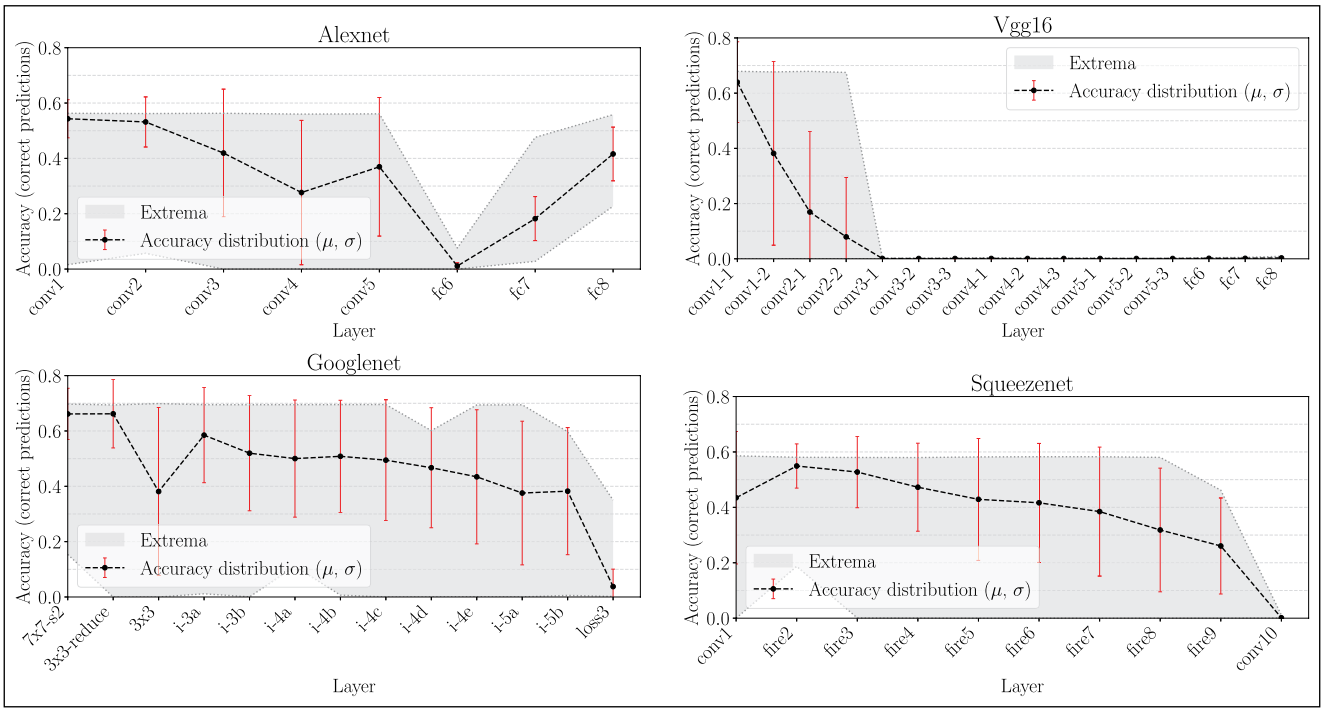

Figure 3. Impact of faults layerwise for the four networks. Each series is represented as the mean top-1 accuracy (black dots), the standard deviation (red error-bars), and the minimum/maximum (gray fill).

accuracy reported is almost always $\approx 0\%$.[6] This means that in some runs, the injected errors were able to fully compromise the network. As rare as it could be, anticipating these cases by studying the network should precede any deployment. Also, the fact that a few number of errors can damage a network this far is another motivation to deeply study the impact of faults.

Discussion

Floating point and fixed point

In contrast to common belief, the $\mathcal{F}$-representation is more vulnerable to injections even though it has more bits. The individual impact of a bit in a short representation (8-bit fixed point) is greater than its counterpart in the $\mathcal{F}$-representation. However, the divergence from the correct value is greater in the latter due to the nature of the representation. The exponent is not represented in the fixed-point representation. A bit-flip in any position is similar to adding or subtracting a power of 2. Since all weights range from -1 to $+1$ [1], the value added or subtracted is minuscule. Hence, its impact can be logically masked.

Bit position

In the $\mathcal{F}$-representation, not all the bits in the exponent are important. The impact of bits is not linear to the bit position but constant except for the most-significant bit, as shown in Figure 2. This is partly due to the distribution of CNN weights. Weights range from -1 to $+1$. High values in the exponent are always accompanied by a negative exponent sign. Having a bit-flip will decrease the value even more, making it close to 0, which is not a big difference considering the range of weights. The only important change is the most-significant bit of the exponent. If changed from 0 to 1, the new value will be orders of magnitude higher than the others. Combined with the maximum pooling, this leads to catastrophic results in the subsequent layers.

Layer index

Although the general tendency shows that the last layers are more vulnerable, no conclusions could be drawn. In other words, vulnerable

[6]The worst case is 0.001 which is equal to randomly guessing the class over the 1000 possibilities.

layers of a new CNN cannot be located without simulation. A fault injection engine such as the one we presented in this article should be used to evaluate the individual layer's vulnerability. This is an extension to Netscope,[7] a neural network visualizer and analyzer. We introduced the resilience parameter, which is computed from the accuracy reported by the fault-injection engine. The modified version allows the extraction of the most vulnerable layers visually. An example output of the analyzer is available with the engine source code.[8]

Suggestions and guidelines

The first study shows a bit difference between the storage formats. The floating-point representation should be used with caution in critical systems. The fixed-point representation would result in less memory and computation[9] overhead with higher reliability. System designers should consider this aspect when dealing with aggressive environments.

It was shown that the most significant bit in the exponent is the vulnerable part of the floating-point representation. Based on this conclusion, the overhead of redundancy techniques could be reduced. Techniques such as triple mode redundancy (TMR) will replicate the whole number three times. This will triple the memory requirements of GoogleNet, for instance, whose number of weights is 6,996,452, thereby adding 427 MB of required storage. If applied exclusively to the vulnerable bits, only 13 MB would be necessary. This result can also be extended to the *layer* test with variable degrees of protection depending on the resilience of each layer as output by the injection engine.

An extensive analysis of the inherent fault tolerance of CNNs is proposed. We show that quantization has a positive impact on reliability. The issue with nonquantized networks is the data representation. For the IEEE-754 format, the most significant bit of the exponent is crucial for the reliability of CNNs. A layer-wise analysis is then performed. For complex networks, the reliability of the overall network should be studied based on the architecture. The framework we developed to analyze reliability is made publicly available. This study is useful in localizing the vulnerable parts of CNNs and helps designing comprehensive low-overhead reliability enhancement techniques.

[7]https://github.com/ethereon/netscope

[8]https://www.github.com/cypox/CNN-Fault-Injector

[9]Fixed point representation is usually coupled with low precision arithmetic. This allows for better efficiency with comparable accuracy.

References

[1] Q. Liu et al., "Security analysis and enhancement of model compressed deep learning systems under adversarial attacks," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASPDAC'18)*, Piscataway, NJ, USA, 2018, pp. 721–726. [Online]. Available: http://dl.acm.org/citation.cfm?id=3201607.3201772

[2] D. S. Phatak and I. Koren, "Fault tolerance of feedforward neural nets for classification tasks," in *Proc. IJCNN Int. Joint Conf. Neural Netw.*, vol. 2, Jun. 1992, pp. 386–391.

[3] Y. Liu et al., "Fault injection attack on deep neural network," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 131–138.

[4] E. B. Tchernev, R. G. Mulvaney, and D. S. Phatak, "Investigating the fault tolerance of neural networks," *Neural Comput.*, vol. 17, no. 7, pp. 1646–1664, Jul. 2005.

[5] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," *arXiv preprint arXiv:1903.12269*, 2019.

[6] P. W. Protzel, D. L. Palumbo, and M. K. Arras, "Performance and fault-tolerance of neural networks for optimization," *IEEE Trans. Neural Netw.*, vol. 4, no. 4, pp. 600–614, Jul. 1993.

[7] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 446–456, Mar. 1995.

[8] J. J. Zhang et al., "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *Proc. IEEE 36th VLSI Test Symp. (VTS)*, 2018, pp. 1–6.

[9] F. dos Santos et al., "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Trans. Rel.*, vol. 68, pp. 663–677, 2018.

[10] M. A. Neggaz et al., "A reliability study on CNNs for critical embedded systems," in *Proc. IEEE 36th Int. Conf. Comput. Design (ICCD)*, 2018, pp. 476–479.

[11] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[12] B. E. Segee and M. J. Carter, "Fault tolerance of pruned multilayer networks," in *Proc. IJCNN-91-Seattle Int. Joint Conf. Neural Netw.*, vol. 2, Jul. 1991, pp. 447–452.