Compact Test Pattern Generation For Multiple Faults In Deep Neural Networks

Dina A. Moussa*, Michael Hefenbrock†, Mehdi Tahoori*
* Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany † RevoAI GmbH, Karlsruhe, Germany
Corresponding Author: Dina A. Moussa (dina.moussa@kit.edu)

***Abstract*—Deep neural networks (DNNs) have achieved record-breaking performance in various applications. To reduce the energy footprint and increase performance, DNNs are often implemented on specific hardware accelerators, such as Tensor Processing Units (TPU) or emerging Memristive technologies. Unfortunately, the presence of various hardware faults can threaten these accelerators' performance and degrade the inference accuracy. This necessitates the development of efficient testing methodologies to unveil hardware faults in DNN accelerators. In this work, we propose a test pattern generation approach to detect fault patterns in DNNs for a common type of hardware fault, namely, faulty weight value representations on the bit level. Contrary to most related works which reveal faults via output deviations, our test patterns are constructed to reveal faults via misclassification which is more realistic for black-box testing.**

I. Introduction

Deep neural networks (DNNs) have succeeded remarkably in many artificial intelligence applications, such as visual recognition, autonomous vehicles, and many more [1]. Since DNNs requires a lot of computational resources, different accelerators such as TPUs [2] or Memristor Crossbar arrays [3] have been developed. However, DNNs implemented on these accelerators are threatened by the existence of faults. Such faults can affect their performance, degrade their inference accuracy and reduce their reliability [4]. Therefore, generating efficient test patterns, to detect these faults, is necessary [5]. This work proposes a test pattern approach that can detect *fault patterns* (instance of multiple faults) in synaptic weight value representations at a bit level.

II. Related works

Most of the related work revealed the presence of faults via output variation [6]–[8], which is impractical for black-box testing, i.e., when there is no access to the model internal parameters, particularly when a trained DNN is considered as an Intellectual Property (IP). Moreover, there is an issue with the instability of measuring the deviation in the presence of normal (non-faulty) runtime variations that can occur during inference operation of DNN accelerators. Also, faults might not be noteworthy if they do not lead to misclassifications, and can safely be ignored.

III. Methodology

A. Fault Modeling

For this work, we consider bit-level fault injection, i.e., altering the bit-level representation of a randomly chosen subset of network parameters. We denote the percentage of altered weights by p_w, and the percentage of their altered bits by p_b. Hence, a combination of (p_w, p_b) determines a specific instance of our fault model. Since in each fault instance multiple weights and multiple bit positions are altered, this can be viewed as a multiple fault model (as opposed to single fault model).

B. Test Pattern Generation

An effective test pattern should reveal faults by producing different labels in the presence of a fault compared to a fault-free model. To encourage this for a test pattern $\mathbf{x}$ in a fault-free network Net versus a network Net_f with a fault f, we adopt the following objective function

$$o(\mathbf{x}, f) := \left\| \mathrm{Softmax}\left(\mathrm{Net}(\mathbf{x})\right) - \mathrm{Softmax}\left(\mathrm{Net}_f(\mathbf{x})\right) \right\|_2^2 .$$

For any practical application, we would want a test pattern to not only detect the specific fault f for which it was generated, but also similar variations of the same fault or fault type. For this purpose, we create a fault list $\mathcal{F}_{(p_w,p_b)} = \{f_n \mid n = 1, \cdots, N\}$ for a given combination of (p_w, p_b). The individual fault f_n thereby denote a specific, randomly generated fault pattern for a faulty neural network instance Net_{f_n}.

Ideally, we would want to generate a set of test patterns that not only detect a single fault, but cover as much of the fault list as possible to encourage a high-overall detection rate. To achieve this, we propose to find test patterns $\mathbf{x}^\star$ as

$$\mathbf{x}^\star = \underset{\mathbf{x}\in\mathbb{X}}{\mathrm{argmax}} \min_{f\in\mathcal{F}} o(\mathbf{x}, f). \tag{1}$$

Hence, $\mathbf{x}$ maximizes the lower bound on the objective $o(\mathbf{x}, f)$ for any f in the fault list. Through this, $\mathbf{x}$ should lead to a misclassification for many $f \in \mathcal{F}$ and thus display a high detection rate. We repeatedly solve (1) until we obtain a set of test pattern that detect all faults in the fault list (see Algorithm 1).

IV. Experiments

The proposed procedure is applied to networks trained on two commonly used benchmark dataset for several configurations of p_w (percentages of weights with bit-flips) and p_b (percentages of flipped bits per weight).

A. Datasets and Experimental Setup

The experimental results were conducted on MNIST and CIFAR-10 with floating fixed point 32 weights implemented in *pytorch* [9]. For MNIST, we use MLP with three fully connected layers of size 20 and LeNet-5 with seven layers, three convolutional layers, and two subsampling layers, followed by two fully connected layers. For CIFAR-10, we use a custom 7-layer neural network ConvNet-7 with four convolutional layers and three fully connected layers.

978-3-9819263-7-8/DATE23/© 2023 EDAA

TABLE I: The overall Experimental results on different models and datasets. The number of fault patterns in the non-targeted list is 1000.

Experimental setup				Generated test patterns	Compaction ratio (∼)	Fault coverage (%)		Number of Iterations	Execution time (∼Sec)
Model and Dataset	p_w (%)	p_b (%)	**Fault patterns**			**Targeted faults**	**Unseen faults**		
MLP (MNIST)	5	100	50	10	5.0x	100	98.6	31	80
	10	70	100	9	11.1x	100	98.7	14	88
	10	50	150	13	11.5x	100	98.2	18	203
LeNet-5 (MNIST)	5	100	50	4	12.5x	100	97.8	6	94
	10	70	150	24	6.2x	100	98.6	127	3411
	10	50	200	31	6.4x	100	98.7	127	8374
ConvNet-7 (CIFAR-10)	5	100	50	2	25.0x	100	99.9	6	327
	10	70	100	10	10.0x	100	99.8	10	1213
	10	50	200	21	9.5x	100	99.5	24	5521

Algorithm 1 The test pattern generation algorithm for a given network Net. The set of found test patterns is stored in $\mathcal{X}$.

```
# Set design parameters and generate fault list
Choose p_w, p_b, N
F ← F_(p_w,p_b) = {f_n | n = 1, ··· , N}
# Define covered/detected fault set for test pattern x
F_c(x) := {f ∈ F | argmax_i Net(x)_i ≠ argmax_i Net_f(x)_i}
X ← ∅ ; k ← 0
while F ≠ ∅ do
    k ← k + 1
    x*_k ← argmax_{x∈X} min_{f∈F} o(x, f)
    F ← F \ F_c(x*_k)
    X ← X ∪ {x*_k}
end while
```

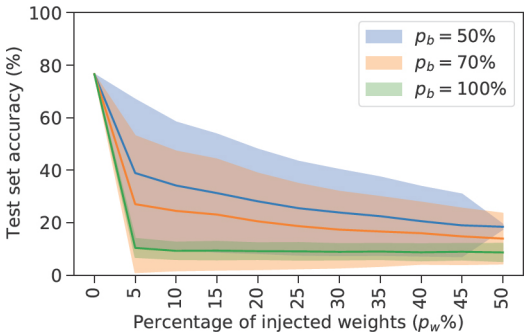

Fig. 1: Accuracy degradation of the ConvNet-7 trained on CIFAR-10 after bit level fault injections.

B. Experimental Results and Discussion

We injected the three prescribed models above with different p_w an p_b with 100 different fault patterns and plotted the resulting mean and standard deviation of the accuracy. The results are displayed in Fig. 1. As can be seen, flipping 50% of the bits in only 5% of the weights can already lead to significant drops in accuracy in the ConvNet-7 model.

We generated different sets of test patterns for each network as explained in Section III. We report results for all networks for various (p_w, p_b) combinations, along with different sizes of the fault list, i.e., $\{50, 100, 150, 200\}$. The result can be found in Table I. Furthermore, the number of iterations needed for the test pattern generation algorithm to cover the faults of the fault list was also reported in Table I. Finally, we also tested our generated test patterns against 1000 faults that have not been part of the fault list. The fault coverage of such faults should be considered the most important evaluation metric as it is an estimate of the expected fault coverage in the field. Here, our test patterns also achieved up to 99.9% fault coverage.

V. Conclusion

This work proposes a compacted test pattern approach that detects multiple weight faults extracted at bit level in the DNNs. In our approach, we maximize the minimum difference between the fault-free and the faulty models that lead to label misclassification. Moreover, an iterative approach is proposed in which we try to detect as many fault patterns as possible with a single test pattern, which leads to compact test pattern set generation. The experimental results show the generated test patterns covered 100% of the targeted fault patterns and achieved a high compaction ratio over different datasets and model architectures. Furthermore, our generated test patterns achieved high fault coverage for untargeted faults, reaching 99.9%.

References

[1] N. Sharma *et al.*, "Machine learning and deep learning applications-a vision," *Global Transitions Proceedings*, vol. 2, no. 1, pp. 24–28, 2021.

[2] A. M. Kist, "Deep learning on edge tpus," *arXiv preprint arXiv:2108.13732*, 2021.

[3] X. Liu *et al.*, "Memristor crossbar architectures for implementing deep neural networks," *Complex & Intelligent Systems*, vol. 8, no. 2, pp. 787–802, 2022.

[4] Y. Ibrahim *et al.*, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[5] C. Torres-Huitzil *et al.*, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017.

[6] Q. Liu *et al.*, "Monitoring the health of emerging neural network accelerators with cost-effective concurrent test," in *2020 57th ACM/IEEE DAC*, 2020, pp. 1–6.

[7] W. Li *et al.*, "Rramedy: Protecting reram-based neural network from permanent and soft faults during its lifetime," *IEEE 37th ICCD*, pp. 91–99, 2019.

[8] H.-Y. Tseng *et al.*, "Machine learning-based test pattern generation for neuromorphic chips," in *IEEE/ACM ICCAD*, 2021, pp. 1–7.

[9] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach *et al.*, Eds., vol. 32. Curran Associates, Inc., 2019.