Exploring Image Selection for Self-Testing in Neural Network Accelerators

Fanruo Meng and Chengmo Yang
Department of Electrical & Computer Engineering
University of Delaware, Newark, USA
{mengfanr, chengmo}@udel.edu

***Abstract*—Hardware accelerators are essential to the accommodation of computation and memory-intensive neural network (NN) applications on resource-constrained edge devices. While hardware accelerators facilitate fast and energy-efficient convolution operations, their accuracy is threatened by various types of faults in their on-chip and off-chip memories, where millions of NN weights are held. To achieve fast and in-time fault detection, a self-test process that periodically runs a small set of test images in the accelerator can be adopted. This paper focuses on developing and comparing multiple numerical score based test image selection strategies. Various image selection criteria are studied, including output probability distribution, gradient sensitivity, and neuron coverage. Experimental studies show that images selected based on the output probability distribution offers high fault detection accuracy over a wide range of fault rates as well as low computation complexity. The small set of test images allows for real-time monitoring of the healthiness of DNN accelerators as well as the subsequent recovery and self-healing process.**

***Index Terms*—DNN Accelerator Reliability, Fault Detection, Self-testing.**

I. Introduction

Deep Neural Network (DNN) has become the go-to solution for many real-world applications, such as face recognition, object detection, disease classification, and self-driving vehicles. Most DNNs maintain a tremendous number of weights and perform intensive convolutional operations during inference, which are computation/memory intensive and energy-hungry. To accommodate DNN applications on resource-constrained edge devices, many DNN accelerators have been developed, including both traditional ASIC and FPGA-based accelerators [1] as well as emerging processing-in-memory (PIM) accelerators [2].

When deployed on a hardware accelerator, the accuracy of the DNN model is challenged by various types of faults in the underlying on-chip and off-chip memories of the accelerator. On one hand, traditional SRAM and DRAM are facing elevated levels of transient and intermittent faults due to their shrinking feature sizes which accentuate the impact of process variation, voltage and temperature fluctuation, and in-progress wear-out. On the other hand, emerging PIM accelerators tend to leverage non-volatile memories (NVM), such as Phase Change Memory (PCM), Resistive RAM (RRAM), and Spin-Transfer Torque RAM (STT-RAM), which also exhibits high levels of faults due to immature fabrication, imprecise programming, process variations, and aging. These faults compromise the reliability of the accelerator memory, result in temporary or persistent variations in the well-trained DNN weights, and noticeably degrade the accuracy of the DNN model if they are not detected and corrected in time.

To tackle the aforementioned fault-induced accuracy drop, it is desirable to constantly monitor the healthiness of the hardware accelerator after deploying the DNN model on it. Unfortunately, this special need cannot be achieved by defect-aware remapping or retraining techniques [3]–[5], which are designed to tolerate permanent detects before deploying the DNN model. Error correction codes (ECC) or checksums which detect and correct errors at bit level are also not preferable, as they add non-trivial overhead of hardware, energy, and timing to the accelerator.

Given the sparsity and inherent robustness of neural networks, it is unlikely for a few faults to noticeably affect accuracy. Therefore, there is no need to detect every single fault in DNN accelerators. Instead, it is more meaningful to detect fault-induced accuracy drop, which can be achieved with image-based self-testing techniques [6]–[8]. These techniques detect accuracy drop with a few test images whose inference results are sensitive to the faults in the underlying accelerator. As the set of test images is small, the computation and storage overhead is trivial. Including more test images can potentially increases test accuracy, but imposes more test overhead. In [8], a comprehensive self-test framework is proposed. Targeting different fault types, multiple sets of test images are down-selected from the test dataset. However, the image selection process has the down-side of being computation-intensive and time-consuming since it relies on two sets of fault injection studies, one to select the most sensitive images and one to evaluate the selected images. The execution time of the image selection process is also unpredictable as it depends on fault types, fault rate, the DNN model, and the dataset. Hence, it can only be executed off-line.

To speed up the image selection process, this paper presents and evaluates a diverse set of image selection strategies that rely on numerical standards rather than statistical information collected from comprehensive fault injection experiments. As a result, they can be integrated into any image-based self-test framework, and is more suitable to large DNNs. Specifically, this work compares five image selection strategies that

combines different metrics such as output probability distribution, gradient sensitivity, and neuron coverage. Comprehensive experimental studies show that images selected with a combination of gradient and output probability offer the highest fault detection accuracy over a wide range of fault rates, while images selected with output probability distribution offer both high detection accuracy and low computation complexity.

The rest of this paper is organized as follows: Section II briefly reviews the related work on DNN accelerators, focusing on fault tolerance and testing. Section III describes and compares the proposed test image selection approaches. Section IV presents the evaluation setup and results, while Section V concludes the paper.

II. Background

A. Impact of Faults on DNN Inference Accuracy

DNN accelerators are susceptible to various types of faults in their on-chip and off-chip memories where millions of DNN weights are held. Traditional SRAM and DRAM suffer from faults and defects caused by Electro-Migration (EM), Time-Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI), Hot-Carrier Injection (HCI), etc [9]. The emerging NVMs used by PIM accelerators also suffer from high levels of faults [10]–[12] due to immature fabrication, imprecise programming, process variations, and aging. For instance, programming of an RRAM cell does not set the conductance to the expected value, but rather on a normal distribution within the objective range [13]. Moreover, process variations make certain cells weaker than the other cells initially, more sensitive to drifting and thermal noise, or more limited by retention time or endurance [14]. These faults can accumulate to a high level, leading to a noticeable drop in DNN model accuracy [15], [16].

While traditionally neural networks are expected to show graceful degradation in their accuracy in the presence of noisy inputs or small variations/errors in the underlying hardware [17], [18], this expectation no longer holds in the face of the elevated fault rates in emerging NVMs. In fact, recent works show that DNN accuracy can be significantly affected by the accumulation of hardware faults [16], [19]–[21]. These faults can alter intermediate computation results or corrupt weight values. While faults in the intermediate results only affect the inference of one input, the impact of faults in weight values is persistent to all inputs.

B. Related Work

Recently researchers started to investigate errors in DNN accelerators. One group of work identifies important weights and relies on remapping and/or retraining process to prevent these weights from being mapped to faulty cells [3]–[5]. A recent work [22] exploits the flexibility in setting the fault-free bits in weight memory to effectively approximate weight values. These techniques are mostly designed to tolerate permanent detects in the accelerator before deploying the DNN model on it. They detect faults by writing a value to a cell and reading it back [23], [24], which destroys the original weight value in the cell and hence cannot be used for detecting transient or intermittent errors when the accelerator is in use.

Another set of previous work generates a small set of testing images that can be utilized for fast fault detection during the inference phase. For example, [6] proposes adversarial example testing that can detect soft errors using a set of adversarial images, generated by adding perturbations to the original image. The work in [7] selects from the original test dataset a set of images with less-confident prediction (based on the logit value of the confidence of output classes), denoted as *corner data*. It also designs a gradient descent based optimization algorithm to generate *white noise-style* test images from scratch. While effective, these methods are dataset-specific and impose extra overhead for generating adversarial images or white noise images.

The self-test framework introduced in [8] presents three test image selection approaches for the purposes of fault detection, fault type identification, and fault rate estimation. It also analyzes the sensitivity (i.e., detection capability) of the selected test sets to different fault types and different fault rates. However, the image selection process requires intensive fault injection studies to down-select the most sensitive images. As the computing overhead of fault injection increases significantly as the model size increases, it can only be executed off-line.

III. Proposed Detection based Image Selecting Framework

This work develops and compares a set of light-weight image selection strategies, each exploiting a certain characteristic of the dataset to form a minimal yet powerful test set to detect fault-induced accuracy drop in DNN accelerators. The main idea is to perform a one-time process to assign a mathematically formulated *score* to each image candidate in the test dataset, which is later used to select the final test set. In the rest of this selection, we present and compare five numerical scores which can be used to rank images to select the desired test set.

A. Corner case based image selection

This strategy is inspired by [7]. Aiming to select images with close top-1 and top-2 classification probabilities, it filters out test images based on their output probability vectors from the softmax layer. The $score_{corner}$ is calculated with Eq.(1), where in is a test image, and $pred()$ is the one dimensional softmax output of the neural network.

$$score_{corner} = max_1(pred(in)) - max_2(pred(in)) \quad (1)$$

The value of $score_{corner}$ measures how close the probabilities of top-1 and top-2 classification results of image in. Images with smaller $score_{corner}$ are more likely to change their classification results from the original label to the second highest label under the influence of faults in the weights.

As observed in our previous work [8], a small distortion in weight values typically causes an image to be misclassified to a higher-ranked label. As the fault rate increases, however, the

image can be misclassified to a lower-ranked class. The higher the fault rate, the more diverse the classification outcome. As a result, images selected with this strategy are expected to show high detection capability under lower fault rates.

B. Gradient based image selection

This strategy aims to select images that are sensitive to faults most probabilistically. It is developed by combining two factors: (1) the expected deviation of weight value under the impact of a single bit-flip, and (2) the partial derivative of the loss function with respect to the weight value, i.e., the gradient. The first factor measures the expected impact of a fault on a weight, while the second measures the impact of weight values on the loss function. For each test image, the product of these two probability values measures how easily the image's lost function changes under the impact of faults.

$$score_{grad} = \sum [ReLU(\underbrace{(-2w-1)/Q}_{A} \times \underbrace{\frac{\partial L}{\partial w}}_{B})] \quad (2)$$

For a Q-bit quantized 2's compliment value, flipping the bit position k increases/decreases the weight value by 2^k if the bit value is 0/1 and $0 \leq k < Q-1$, while flipping the most significant bit increases/decreases the weight value by 2^{Q-1} if the bit value is 1/0. Assuming each bit has equal probability to be flipped, the expected deviation of the weight value under the impact of a single bit-flip is $(-2w-1)/Q$, with w denoting the weight value and Q the quantization level. Portion B of Eq. (2) computes the gradient. For each image, Eq. (2) sums up the product of these two probability values across all the weights. Images with higher $score_{grad}$ are more likely to be influenced by faults in the weights of the NN model.

C. Corner-case and gradient based image selection

This strategy combines the advantages of $score_{corner}$ and $score_{grad}$. The former measures how easily an image may change its classification, while the latter measures the sensitivity of the loss function to random faults. By combining these two, this strategy aims to select images that are most likely to change classification results under the impact of random faults. Specifically, $score_{corner/grad}$ is calculated with Eq. (3). Regularity 1 is added in the denominator to keep the value of $score_{corner/grad}$ within certain range. Images with highest scores are selected as test images.

$$score_{corner/grad} = score_{grad}/(1 + score_{corner}) \quad (3)$$

D. Probability distribution based image selection

Unlike the corner case strategy that measures the difference between top-1 and top-2 output probabilities, this strategy measures the sum of remaining output probabilities other than the top-1 class. It is based on the observation that as the fault rate increases, the distribution of output class ranking moves more towards the lower end, indicating more confusion in the faulty NN [8]. In other words, images with more flatten output probability distribution are more likely to be affected by faults. The ranking score is defined in Eq. (4).

$$score_{prob\_distrib} = 1 - max(pred(in)) \quad (4)$$

When applied to standard softmax output vectors, Eq. (1) measures the sum of remaining output probabilities other than the top-1 class. However, when the weights are quantized to Q-bit integers, elements in the output probabilities do not always add up to 1 and some elements may be negative. With this in mind, we develop Eq. (5) which ignores negative elements in the output vector with the ReLU activation function (which keeps positive values intact while forcing negative values to zero). Images with larger scores have more flatten probability distribution, and therefore are selected as test images.

$$score_{prob\_distrib} = 1 - \frac{max(pred(in))}{sum(ReLU(pred(in))} \quad (5)$$

E. Coverage based image selection

This strategy is inspired by the work in [25]. The activation function plays an important role in fault propagation. If a neuron is not activated by the input image, faults in it are masked. Since the activation status of neurons varies across different inputs, the goal of this strategy is to select images which activate most neurons as test vectors, as faults are less likely to be masked and more likely to propagate to the output layer and affect classification outcome.

To calculate the activation status of a given image in for a specific layer with C input channels and K output features, we use a K-bit binary vector, with each bit i indicating the activation status of neuron i. $H[x]$ is a step function ($H[x] = 0$ if $x \leq 0$ and $H[x] = 1$ if otherwise).

$$activation(i, in) = H[\sum_{c=1}^{C} ReLU(output_i(in, c))] \quad (6)$$

After obtaining the activation status of each neuron for each test image, the goal is to select a minimum set of test images that cover all the neurons in all layers. This is a minimum set cover problem, which can be solved with a greedy heuristic, shown in Algorithm 1. In lines 3-5, the algorithm selects all the images (i.e., rows) that uniquely activate certain neurons (i.e., columns). Then, in lines 6-13, it uses a loop to select, at every iteration, the image that covers the maximum number of uncovered neurons. If multiple images cover the same number of remaining neurons, it selects the image with highest activation scores across all neurons and all layers (line 8). $score_{activation}$ can be computed with Eq. (6), where N refers to the total number of neurons in the NN. It is not a binary vector but a numerical number that measures how sensitive a neuron is to the input image.

$$score_{activation} = \sum_{i=1}^{N} \sum_{c=1}^{C} ReLU(output_i(in, c)) \quad (7)$$

Algorithm 1 Filter activation coverage based image selection Algorithm

Input:

$M_{input}(X, N) \leftarrow$ the binary activation vectors of all the input images, $X \leftarrow$ number of images in the test dataset. $N \leftarrow$ total number of neurons.

$score_{activation}$ of all input images.

Output: Selected image set Y;

Choose all the images that uniquely activate certain neurons and add them into Y;

Delete the rows in $M_{input}(X, N)$ corresponding to these images;

Delete the columns in $M_{input}(X, F)$ (corresponding to the neurons) that are already covered;

while $M_{input}(X, N)$ has 1's **do**

    Calculate the sum of 1s on each row and choose the row with the maximum sum;

    if $\#ofcandidates > 1$ **then** use their $score_{activation}$ to break the tie;

    end if

    Add the image selected into Y;

    Delete the corresponding row in $M_{input}(X, N)$;

    Delete the columns in $M_{input}(X, N)$ covered by the selected row;

end while

F. Image Selection Complexity

The image selection strategies described in this section are model specific, as they require both the test data set and the trained DNN model as inputs. However, the selection process depends neither on the fault type nor on fault rate. Hence, the scores of each test image only need to be calculated once, and no fault injection studies are needed for test image selection.

Table I summarizes the image selection complexity of the proposed strategies. Among them, the corner case and probability distribution based methods have the lowest complexity as these scores are calculated purely based on the softmax output of each image. The coverage based strategy has more computation overhead, as it needs to pull out the activation values of each neuron and solves a minimum set cover problem. The two methods that require gradient information also involve more computation, as the gradient for each weight needs to be calculated for each test image. Finally, the sensitivity score used in [8] imposes the highest computation overhead, as it requires intensive fault injection studies to evaluate every test image under different fault rates and fault types.

IV. Experimental Evaluation

A. Experimental Setup

The different test image selection strategies are evaluated on the CIFAR-10 [26] dataset for two network structures, the VGG-16 [27] and ResNet-20 [28]. CIFAR-10 is an object recognition dataset containing 60K (50K training and 10K testing) 32x32 color images, with a total of 10 classes and 6K images per class. Table II lists the layer count, weight count, and accuracy of the two network structures studied in this work. VGG-16 is a typical DNN structure for image recognition composed of 13 convolutional layers and 3 fully-connected layers. It achieves the highest accuracy of 90.87% after 50 epochs of training. ResNet-20 employs residual blocks to avoid gradient exploring and vanishing, and achieves 91.49% accuracy after 80 epochs of training. Since most NN accelerators utilize quantized weight values to reduce the model size and eliminate expensive floating-point arithmetic operations, we quantize the weights of both models to 8-bit 2's compliment numbers.

TABLE I: Complexity of different image selection methods

Strategy	Selection complexity
$score_{grad}$	medium
$score_{corner}$	low
$score_{corner/grad}$	medium
$score_{probdistrib}$	low
$score_{activation}$	medium
$score_{sensitivity}$ [8]	high

TABLE II: Properties of evaluated DNN models

DNN	Layers	Weights	Accuracy
VGG-16	16	1.5×10^7	90.87%
ResNet-20	20	2.7×10^5	91.49%

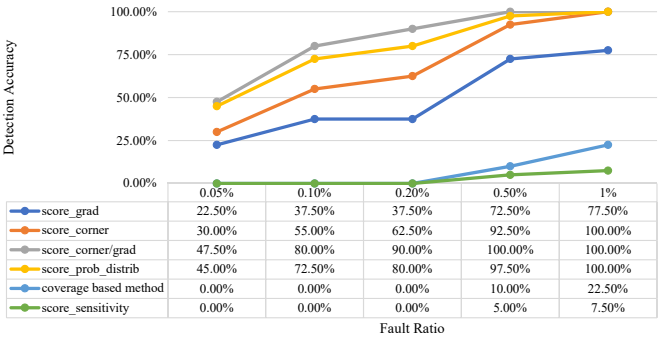
The proposed image selection strategies are evaluated under two representative fault models:

- *Random*: These faults model the impact of transient or permanent faults on the NN accelerators, by randomly flipping certain bits of weight values according to a predefined fault rate. Each bit of each weight is given an equal rate to be flipped.
- *Worst-case*: This fault model randomly selects weights according to a predefined fault rate, and flips the most significant bit (MSB) of the selected weights. It is specifically designed to model malicious reliability attacks [29], [30] where the attacker intentionally triggers the biggest impact by flipping the MSB of selected weight values.

All fault injection studies as well as the accuracy assessment are performed in Pytorch [31]. We have evaluated a total of 400 fault maps (200 for random faults and 200 for worst-case faults) for both VGG-16 and ResNet-20 under five fault rates of 0.05%, 0.1%, 0.2%, 0.5% and 1%. Faults are injected in all layers of the two models. The test sets are generated by sorting images based on their scores (in the ascending order of $score_{corner}$ and in the descending order of other scores). Based on the study of image set size in [8], we select 20 images for each strategy except for the coverage based method, which requires 18 images to cover all the neurons in VGG-16 and 12 images to cover all the neurons in ResNet-20.

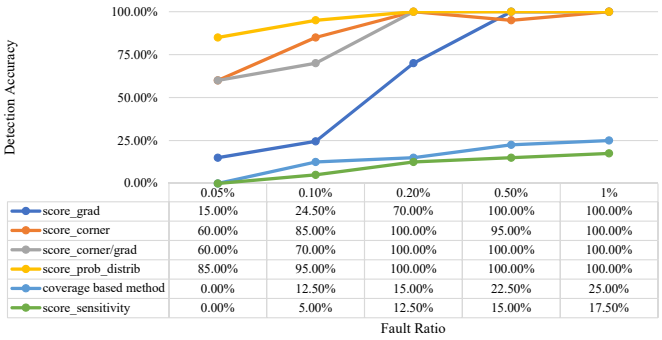
B. Evaluation Results

This section compares the fault detection capability of the different image selection strategies and provides an in-depth analysis of their behavior in the face of different fault types.



	0.05%	0.10%	0.20%	0.50%	1%
score_grad	22.50%	37.50%	37.50%	72.50%	77.50%
score_corner	30.00%	55.00%	62.50%	92.50%	100.00%
score_corner/grad	47.50%	80.00%	90.00%	100.00%	100.00%
score_prob_distrib	45.00%	72.50%	80.00%	97.50%	100.00%
coverage based method	0.00%	0.00%	0.00%	10.00%	22.50%
score_sensitivity	0.00%	0.00%	0.00%	5.00%	7.50%

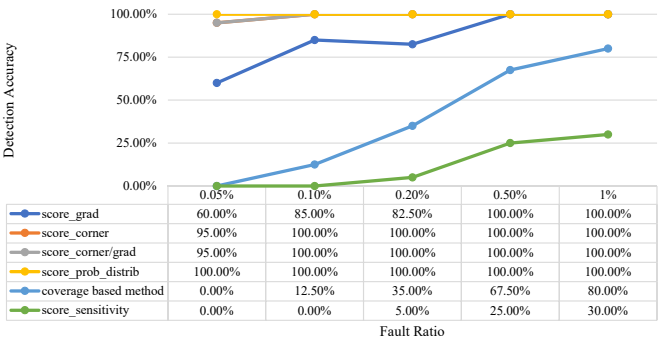
(a) VGG-16



	0.05%	0.10%	0.20%	0.50%	1%
score_grad	15.00%	24.50%	70.00%	100.00%	100.00%
score_corner	60.00%	85.00%	100.00%	95.00%	100.00%
score_corner/grad	60.00%	70.00%	100.00%	100.00%	100.00%
score_prob_distrib	85.00%	95.00%	100.00%	100.00%	100.00%
coverage based method	0.00%	12.50%	15.00%	22.50%	25.00%
score_sensitivity	0.00%	5.00%	12.50%	15.00%	17.50%

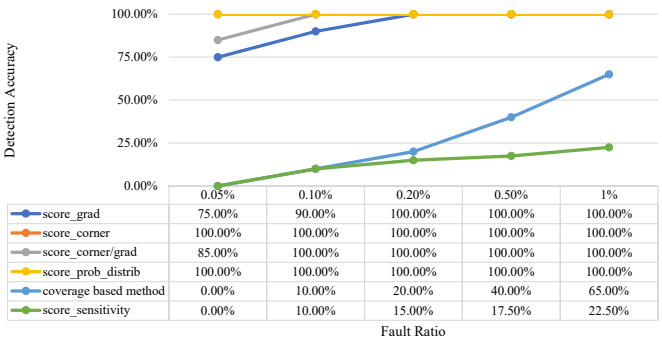
(b) ResNet-20

Fig. 1: Detect accuracy of the images selected with the proposed methods and sensitivity scores [8] on random faults



	0.05%	0.10%	0.20%	0.50%	1%
score_grad	60.00%	85.00%	82.50%	100.00%	100.00%
score_corner	95.00%	100.00%	100.00%	100.00%	100.00%
score_corner/grad	95.00%	100.00%	100.00%	100.00%	100.00%
score_prob_distrib	100.00%	100.00%	100.00%	100.00%	100.00%
coverage based method	0.00%	12.50%	35.00%	67.50%	80.00%
score_sensitivity	0.00%	0.00%	5.00%	25.00%	30.00%

(a) VGG-16



	0.05%	0.10%	0.20%	0.50%	1%
score_grad	75.00%	90.00%	100.00%	100.00%	100.00%
score_corner	100.00%	100.00%	100.00%	100.00%	100.00%
score_corner/grad	85.00%	100.00%	100.00%	100.00%	100.00%
score_prob_distrib	100.00%	100.00%	100.00%	100.00%	100.00%
coverage based method	0.00%	10.00%	20.00%	40.00%	65.00%
score_sensitivity	0.00%	10.00%	15.00%	17.50%	22.50%

(b) ResNet-20

Fig. 2: Detect accuracy of the images selected with the proposed methods and sensitivity scores [8] on MSB faults

1) Random faults: Fig. 1(a) and 1(b) compare the detection capability of different image selection strategies on random faults. For both VGG-16 and ResNet-20, the coverage based image selection method has relatively low detection accuracy across all fault rates compared with other approaches. The reason for that is because this method focuses on the activation status of the neurons and does not consider the relative importance of each neuron. Since the optimization goal is to achieve the maximum coverage, it may end up selecting images that uniquely activate certain unimportant neurons.

The corner case based and gradient based image selection methods perform much better than the coverage based method for VGG-16. The combination of these scores can filter out more sensitive images that together achieve higher detection accuracy. In Fig. 1(a), when fault rate is 0.05%, $score_{corner/grad}$ can achieve at least 17.5% higher accuracy than $score_{grad}$ and $score_{corner}$ and as the fault rate increases, its performance boosts dramatically and the gap between it and $score_{grad}$ or $score_{corner}$ keeps increasing. The trend in ResNet-20 is slightly different. While $score_{corner}$ offers high detection accuracy across all tested fault rates, $score_{grad}$ is not quite sensitive to faults under low fault rates (0.1% – 0.2%). This also affects the performance of $score_{corner/grad}$ under low fault rates. Regarding the probability distribution based method, for VGG-16 it is noticeable that $score_{prob\_distrib}$ has much better performance than $score_{grad}$ and $score_{corner}$ but slightly worse than $score_{corner/grad}$. For ResNet-20, $score_{prob\_distrib}$ outperforms all the other methods, as the shallower and deeper network topology flattens the output distribution. If we take the computing complexity into consideration, $score_{prob\_distrib}$ is the best choice since it only needs to compute the output probability vector, whereas $score_{corner/grad}$ needs intensive computation to calculate the gradient of each weight for each image.

Finally, the set of images selected based on sensitivity scores [8] do not perform well. This is because the fault injection experiments performed for image selection in [8] were done in Keras [32], which uses a different quantization approach from Pytorch, the experimental platform used in this paper. This also shows that the statistical based image selection method is dedicated to a specific quantized model, and cannot be generalized to different platforms or quantization processes.

Overall, these experiments show that the probability distribution based method is the best strategy by offering both high detection accuracy and low computing computation complexity for image selection.

2) MSB faults: The detection accuracy of the selected images for worst-case MSB faults are reported in Fig. 2(a) and Fig. 2(b) for VGG-16 and ResNet-20, respectively. As expected, all the sets of images offer much higher detection ac-

curacy on these worst-case faults. The different strategies still follow the same trend. The $score_{corner}$, $score_{corner/grad}$ and $score_{prob\_distrib}$ are the best selection strategies. Among them, the probability distribution based method ($score_{prob\_distrib}$) offers highest detection accuracy under the lowest fault rate 0.05%. The coverage based method also offers much higher detection accuracy under fault rates 0.5%–1% since MSB faults cause more severe and noticeable accuracy drop than random faults.

V. Conclusion

In this work, we presented five light-weight test image selection strategies that can be employed by an image-based self-test framework to monitor the healthiness of neural network hardware accelerators. These strategies select images with numerical scores, which are more general and less computation-intensive compared to the previous fault injection-based image selection work [8]. Experimental studies show that the score combining corner-case and gradient ($score_{corner/grad}$) offers the highest overall fault detection performance for both random and MSB faults, while the probability distribution based method ($score_{prob\_distrib}$) is a better choice by offering both high detection accuracy and low computing complexity.

Acknowledgments

This work is supported by Semiconductor Research Corporation grant #2964.001 and National Science Foundation grant #1909854.

References

[1] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015.

[2] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, 2016, pp. 14–26.

[3] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *ACM/EDAC/IEEE Design Automation Conference*, 2017, pp. 1–6.

[4] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Design, Automation Test in Europe Conference Exhibition*, March 2017, pp. 19–24.

[5] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems," in *ACM/EDAC/IEEE Design Automation Conference*, 2017, pp. 1–6.

[6] W. Li, W. Wang, H. Li, and X. Li, "RRAMedy: Protecting ReRAM-based neural network from permanent and soft faults during its lifetime," in *IEEE International Conference on Computer Design*, 2019, pp. 91–99.

[7] Q. Liu, W. Wen, and C. Yang, "Monitoring the health of emerging neural network accelerators with cost-effective concurrent test," in *ACM/IEEE Design Automation Conference*, 2020, pp. 91–99.

[8] F. Meng, F. S. Hosseini, and C. Yang, "A self-test framework for detecting fault-induced accuracy drop in neural network accelerators," in *Asia and South Pacific Design Automation Conference*, 2021, pp. 722–727.

[9] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, 2003.

[10] C. Chen, H. Shih, C. Wu, C. Lin, P. Chiu, S. Sheu, and F. T. Chen, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 180–190, 2015.

[11] B. Li, Y. Wang, Y. Chen, H. H. Li, and H. Yang, "ICE: Inline calibration for memristor crossbar-based computing engine," in *Design, Automation Test in Europe Conference Exhibition*, 2014, pp. 1–4.

[12] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *IEEE International Symposium on High Performance Computer Architecture*, 2015, pp. 476–488.

[13] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[14] B. Feinberg, S. Wang, and E. Ipek, "Making memristive neural network accelerators reliable," *IEEE International Symposium on High Performance Computer Architecture*, pp. 52–65, 2018.

[15] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based reconfigurable in-situ accelerator," in *Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 288–301.

[16] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal brain damage: exposing the graceless degradation in deep neural networks under hardware fault attacks," in *USENIX Security Symposium*, 2019, pp. 497–514.

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

[18] C. H. Sequin and R. D. Clay, "Fault tolerance in artificial neural networks," in *International Joint Conference on Neural Networks*, 1990, pp. 703–708 vol.1.

[19] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems," in *ACM/EDAC/IEEE Design Automation Conference*, 2017, pp. 1–6.

[20] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1211–1220.

[21] F. Yao, A. S. Rakin, and D. Fan, "DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *USENIX Security Symposium*, Aug. 2020, pp. 1463–1480.

[22] F. S. Hosseini, F. Meng, C. Yang, W. Wen, and R. Cammarota, "Tolerating defects in low-power neural network accelerators via retraining-free weight approximation," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 5s, 2021.

[23] A. J. Van de Goor and Y. Zorian, "Effective march algorithms for testing single-order addressed memories," *Journal of Electronic Testing*, vol. 5, no. 4, pp. 337–345, 1994.

[24] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu, "Modeling, detection, and diagnosis of faults in multilevel memristor memories," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 5, pp. 822–834, 2015.

[25] Z. He, T. Zhang, and R. Lee, "Sensitive-sample fingerprinting of deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[26] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[29] A. Sreedhar, A. Kundu, and I. Koren, "On reliability trojan injection and detection," *Journal of Low Power Electronics*, vol. 8, pp. 674–683, 2012.

[30] C. Cook, S. Sadiqbatcha, Z. Sun, and S. X.-D. Tan, "Reliability based hardware trojan design using physics-based electromigration models," in *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, 2018, pp. 5–8.

[31] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[32] F. Chollet *et al.*, "Keras," https://keras.io, 2015.