A Self-Test Framework for Detecting Fault-induced Accuracy Drop in Neural Network Accelerators

Fanruo Meng, Fateme S. Hosseini, Chengmo Yang
Electrical and Computer Engineering, University of Delaware
{mengfanr,fateme,chengmo}@udel.edu

ABSTRACT

Hardware accelerators built with SRAM or emerging memory devices are essential to the accommodation of the ever-increasing Deep Neural Network (DNN) workloads on resource-constrained devices. After deployment, however, the performance of these accelerators is threatened by the faults in their on-chip and off-chip memories where millions of DNN weights are held. Different types of faults may exist depending on the underlying memory technology, degrading inference accuracy. To tackle this challenge, this paper proposes an online self-test framework that monitors the accuracy of the accelerator with a small set of test images selected from the test dataset. Upon detecting a noticeable level of accuracy drop, the framework uses additional test images to identify the corresponding fault type and predict the severeness of faults by analyzing the change in the ranking of the test images. Experimental results show that our method can quickly detect the fault status of a DNN accelerator and provide accurate fault type and fault severeness information, allowing for subsequent recovery and self-healing process.

CCS CONCEPTS

• **Hardware → Self-checking mechanisms**; **Error detection and error correction**; *System-level fault tolerance.*

KEYWORDS

Neural Network Accelerator, Fault tolerance, Reliability, Testing

ACM Reference Format:
Fanruo Meng, Fateme S. Hosseini, Chengmo Yang. 2021. A Self-Test Framework for Detecting Fault-induced Accuracy Drop in Neural Network Accelerators. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431519

1 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 [23] as well as emerging processing-in-memory (PIM) accelerators [18].

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. On one hand, traditional SRAM and DRAM are facing elevated levels of transient and intermittent faults due to their shrinking feature size which accentuates 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 exhibit high levels of faults due to immature fabrication, imprecise programming, process variations, and aging. These faults together result in temporary or persistent variations in the well-trained DNN weights, which noticeably degrade the accuracy of the DNN model if they cannot be detected and corrected in time.

To tackle the aforementioned fault-induced accuracy drop, it is desirable to monitor the health of the hardware accelerator after deploying the DNN model on it. Unfortunately, this cannot be achieved by defect-aware remapping or retraining [3, 15, 21], which are designed to tolerate permanent detects before deploying the DNN model. Error correction codes (ECC) or checksums are also not preferable, as they add non-trivial overhead of hardware, energy, and timing to the accelerator. To reduce such overhead, one possibility is to develop a few test inputs whose inference results are sensitive to the faults in the underlying accelerator [14, 16]. Including more test inputs can potentially increases test accuracy, but imposes more test overhead. However, previous work [14, 16] fall short of analyzing the impact of test size and the sensitivity of these test images to different fault types and rates.

Unlike previous work which only checks whether an accelerator is faulty or not [14, 16], in this work we propose a comprehensive *self-test framework* which adopts a two-stage test and diagnosis process and offers three functions: fault detection, fault type identification, and fault severeness prediction. We propose three approaches that select different test images for these functions. For fault detection, we select images that are generally sensitive to different types of faults, whereas for fault type identification, we select images most sensitive to a specific type of fault. For fault severeness prediction, we select images whose output ranking is sensitive to the fault rate. We perform comprehensive experimental studies on the quality of the three types of test images, and further examine the impact of test data size on test accuracy and test overhead. The

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431519

proposed self-test framework is well suited to resource-constrained accelerators as it requires no hardware support, imposes minimum test overhead, and targets the most representative quantization levels (8-16 bits) of DNN accelerators.

The rest of this paper is organized as follows: Section 2 briefly reviews the related work on DNN accelerators, focusing on fault tolerance and testing. Section 3 describes the proposed self-test framework as well as the three test image selection approaches. Section 4 presents the evaluation setup and results, while Section 5 concludes the paper.

2 RELATED WORK

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 [5]. The emerging NVMs used by PIM accelerators also suffer from high levels of faults [2, 12, 22] 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 [1]. Moreover, process variations make certain cells weaker than the others initially, more sensitive to drifting and thermal noise, or more limited by retention time or endurance [6]. These faults can accumulate to a high level, leading to a noticeable drop in DNN model accuracy [8, 13].

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, 15, 21]. These techniques are mostly designed to tolerate permanent detects in the accelerator before deploying the DNN model onto it. They detect faults by writing a value to a cell and reading it back [10, 20], 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, [14] proposes adversarial example testing that can detect soft errors using a set of adversarial images, generated by adding perturbations to the original image. While effective, this method is dataset-specific and imposes extra overhead for generating adversarial images. The work most related to this work is [16], which 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*. However, the work does not analyze the sensitivity (i.e., detection capability) of the selected test set to different fault types and different fault rates. In comparison, this work presents three test image selection approaches for different purposes of fault detection, fault type identification, and fault rate estimation. Our work also needs a much smaller test set than [16], making it more suitable to resource-constrained accelerators.

Compared to fault detection, fault rate estimation is a much more challenging task. In [7], it is observed that in a faulty network, not only the accuracy drops, but many test images are misclassified as the same label. A label-stuck method is then proposed to estimate the fault rates. Our work differs from [7] in that it checks not the labels but the ranking of the output classes to determine how severe the faults are and how urgent the recovery is needed.

3 PROPOSED SELF-TEST FRAMEWORK

This section first presents a functional overview of the proposed self-test framework, then explains the three different test image selection approaches, and finally gives the fault detection, fault type identification, and fault severeness prediction algorithms.

3.1 Framework Overview

The proposed self-test framework leverages the fundamental inference capability of a neural network accelerator. It employs a small set of test images whose inference results of the fault-free network are known *a priori*. By running these images through the accelerator, its healthiness can be determined quickly.

Fig. 1 presents an overview of the proposed framework, composed of two major parts, namely, *test image selection* and *self-test procedure*. Test image selection is performed once the network is trained and is carefully designed to down select the network's original test images to generate three small test sets, namely, *general*, *fault type specific*, and *fault rate sensitive* test images. These test sets are stored in the accelerator to be used during the self-test procedure. To minimize test overhead, each set only contains a very small number of images. The self-test procedure is periodically activated when the accelerator is in use, i.e., after deploying the well-trained neural network model. As Fig. 1 shows, the procedure leverages the general test images to perform fault detection. If the top-1 classification result of any image changes, the accelerator is considered faulty and the following two functions are activated to obtain more specific fault information: *Fault type identification* aims to predict the most possible fault type (among K target types), while *Fault severeness prediction* compares and analyzes the ranking of image output classes in the fault-free and faulty models to predict fault severeness.

The output of the self-test procedure indicates whether the accelerator is faulty and predicts the fault type and severeness. As Fig. 1 shows, a fault-free accelerator continues its normal operation until the next test cycle, while a faulty accelerator utilizes the predicted fault type and severeness to invoke a proper recovery plan. Such information is vital since not all types/severeness of faults are fixed in the same way. For example, some types of faults can only be fixed by writing the correct value back to the memory cell while the negative impact of other faults (e.g., resistance drifting in PCM and RRAM) can be mitigated via adding/subtracting a value from the computation outcome. In terms of fault severeness, a relatively healthy accelerator (with low fault rates) can go through an error correction procedure leveraging ECCs in the memory (if available) to ensure its correct operation, while an accelerator whose fault rate exceeds ECC capability would demand reprogramming of all its memory cells.

3.2 Test Image Selection

To minimize test overhead, the self-test procedure is designed to be relatively straightforward and the test set size is kept as small

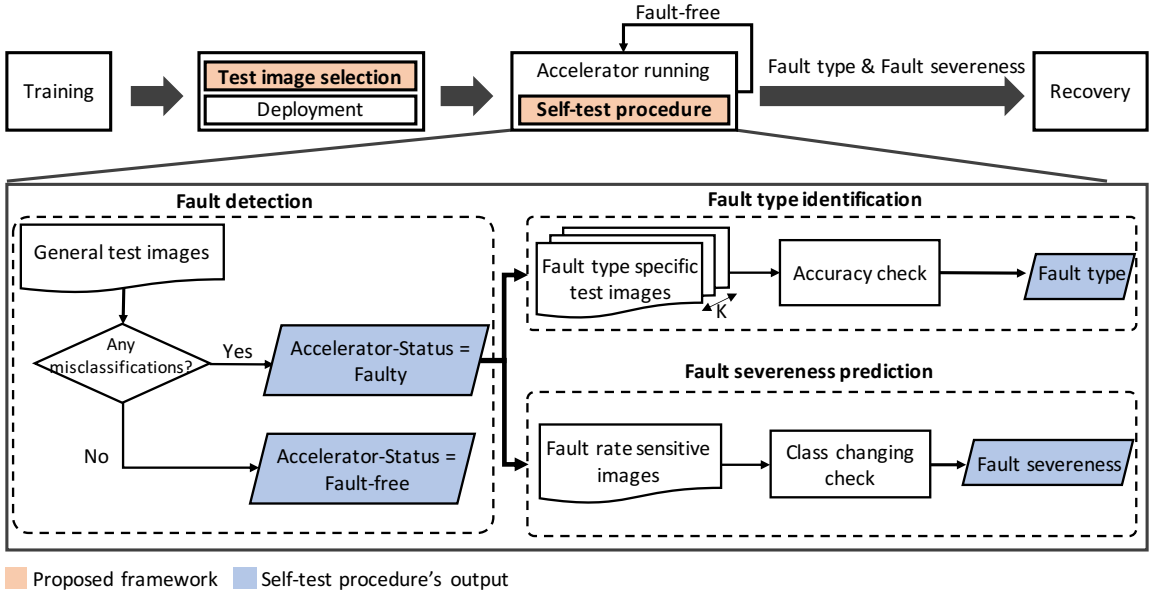


Figure 1: Overview of proposed self-test fault detection and diagnosis framework

as possible. The accuracy of the self-test procedure depends on the quality of the test images. To select the most suitable images for each of the three test functions, our framework follows a set of different rules and selects test images in three steps: (1) mimicking the behavior of faulty accelerators with a set of synthetic fault maps that are randomly generated following different combinations of fault models and fault rates; (2) evaluating the original test set (e.g., 10,000 images) on these faulty accelerators and collecting their classification results, (3) interpreting the classification outcome by introducing and utilizing the notion of *sensitivity*.

Definition 3.1. Given a trained neural network model with T images in the original test dataset, the sensitivity of test image I to fault type f is formulated as:

$$(S_f)_I = N_{dev}/N_{maps} \tag{1}$$

where N_{maps} is the number of synthetic fault maps generated[1] and N_{dev} is the number of times that image I changes its classification results (from the fault-free network). Since $N_{dev} \leq N_{maps}$, we have $0 \leq (S_f)_I \leq 1$.

General test images: As the primary goal of this set is to quickly check whether the accelerator under test is faulty or not, it should contain a minimum set of images that are sensitive to all K different fault types. To this end, we define a General Score (GS) to quantitatively measure the general sensitivity of an test image I from the original test dataset T:

$$GS_I = \sum_{f=1}^{K}(S_f)_I - \sigma_I \tag{2}$$

[1]If the accelerator is exposed to K potential fault types and n synthetic fault maps are randomly generated for each fault type, then we have $N_{maps} = K \times n$.

where σ_I is the standard deviation of the sensitivity values of image I to different fault types, as shown below:

$$\sigma_I = \sqrt{\frac{1}{K-1}\sum_{f=1}^{K}|(S_f)_I - \mu|^2}, \quad \mu = \frac{1}{K}\sum_{f=1}^{K}(S_f)_I \tag{3}$$

By subtracting σ_I, the GS score favors images that have similar high sensitivity scores across all fault types. The general test image set is created by selecting images with the highest GS scores from the original test dataset T.

Fault type specific test images: The primary goal of this set is to pinpoint the most likely fault type out of K possible candidates. Again, the image selection process is based on the sensitivity scores. An image I suitable for pinpointing fault f should have a high sensitivity score $(S_f)_I$ but low scores $(S_j)_I$ for any other fault type j. Specifically, we use two thresholds to filter out suitable images: Th_f and Th_j. An image that is *sensitive* to fault f and *insensitive* to fault j should have $S_f \geq Th_f$ and $S_j \leq Th_j$. The images that fulfill these two conditions can be ranked based on a Type Specific Score (TSS):

$$(TSS_I)\_f = \frac{(S_f)_I - Th_f}{1 - Th_f} + \frac{\sum_{j \neq f}(1 - (S_j)_I/Th_j)}{K-1} \tag{4}$$

The first part of the TSS score measures the sensitivity of image I to fault type f. Since $Th_f \leq (S_f)_I \leq 1$, the value of the first part is in the range of $[0, 1]$. The higher the $(S_f)_I$, the higher the TSS score. In comparison, the second part of Eq. (4) measures the sensitivity of image I to the other fault types. Since $0 \leq (S_j)_I \leq Th_j$, the value of the second part is also in the range of $[0, 1]$. The lower the $(S_j)_I$, the higher the TSS score. By combining the two parts, Eq. (4) is effective in filtering out test images specific to a given fault type, i.e., those with the highest TSS scores.

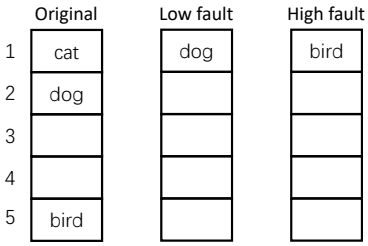


Figure 2: Impact of fault rate on output ranking change

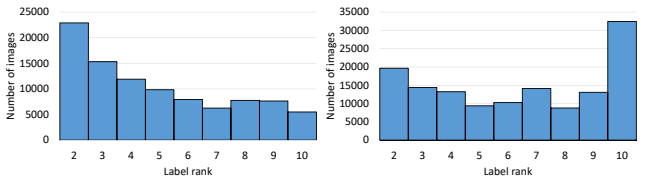
The size and quality of the fault type specific test sets are largely affected by the two thresholds Th_f and Th_j. While the ideal case desires higher Th_f and lower Th_j values, in reality the sensitivity scores $(S_f)_I$ and $(S_j)_I$ are usually positively correlated and hence, the two thresholds should be adjusted in the same direction. Increasing Th_f and Th_j will make the test set more specific, while decreasing Th_f and Th_j will make the set more sensitive[2].

Ideally, for K possible fault types, K subsets of test images should be filtered out, each containing images that are only sensitive to a specific fault type. Our study shows that this is feasible for most fault types except for random faults. As random faults have no particularity, images that are sensitive to random faults are also sensitive to other faults. As a result, our framework generates $K-1$ sets of test images each targeting non-random types of faults. If no specific fault type is detected, the fault type is classified as random.

Fault rate sensitive images: When the fault rate in the accelerator increases, it is expected that more images will be misclassified. However, it is extremely costly and time-consuming to store the original test set T in the accelerator and run each test image to measure the fault rate. Instead, we develop a test image down selection approach based on *rank distribution diversity check*. The observation is that a higher fault rate causes not only more severe accuracy drop but also a more random guess among the possible output classes. The change in output class rank is illustrated in Fig. 2. An image labeled 'cat' is classified correctly in the fault-free network but misclassified in the faulty networks. A small distortion in weight values typically causes the image to be misclassified to an originally higher-ranked label (e.g., 'dog'). As the fault rate increases, however, the image can be misclassified into a lower-ranked class (e.g., 'bird'). The higher the fault rate, the more diverse the classification outcome. To illustrate such correlation, Fig. 3 presents the deviation in output class ranking of all the test images of CIFAR-10 dataset, with 0.1% and 2% of random bit flips injected into the NN model (16-bits). At a higher fault rate, the distribution of ranking moves more towards the lower end, indicating that the faulty network is more confused.

To filter out test images, we generate synthetic fault maps following two different fault rates, *lowF* and *highF* (e.g., 0.1% vs 2%). For each image, its top-1 label in the faulty network is found and the corresponding rank in the original network is collected. An image I is sensitive to fault rate changes if its rank $(R_{lowF})_I$ is less than a threshold Th_l and its rank $((R_{highF})_I$ is larger than a threshold Th_h.

[2] *Test sensitivity* is the ability to correctly identify those NN models with the fault type (true positive rate), whereas *test specificity* is the ability to correctly identify those without the fault type (true negative rate).



(a) 0.1% random faults (b) 2% random faults

Figure 3: Rank distribution of faulty outputs in the original Cifar-Net classification. Column r shows the number of images classified by the faulty network as a particular class whose rank in the original network is r.

Algorithm 1 Self-Test Procedure

```
1:  Apply general test images
2:  if classification result of any image changed then
3:      Accelerator is faulty
4:      for all f = 1 to K − 1 (fault types) do
5:          Apply test image of fault type f
6:          if classification of t_f% or more images changed then
7:              add f to accelerator major fault type
8:      if accelerator major fault type is ϕ then
9:          accelerator major fault type = default (random)
10:     Apply fault rate sensitive test images
11:     if Severe Score is higher than s_high then
12:         Fault severeness = high
13:     else if Severe Score is lower than s_low then
14:         Fault severeness = low
15:     else
16:         Fault severeness = medium
17: else
18:     Accelerator is fault-free
```

The images that fulfill these two conditions can be ranked based on a Rate Sensitive Score *RSS*:

$$RSS_I = [Th_l - (R_{lowF})_I] + [(R_{highF})_I - Th_h] \quad (5)$$

For a given fault rate, typically n different synthetic fault maps are randomly generated. The scores $(R_{lowF})_I$ and $(R_{highF})_I$ are the average scores across all n fault maps.

3.3 Self-test Procedure

With the three sets of test images selected following the approaches described above, the proposed self-test procedure can be conducted in three steps, shown in Algorithm 1. The algorithm begins with evaluating the general test images on the accelerator. If the classification result of any of these images changes, the accelerator is marked faulty, and the fault type identification and fault severeness prediction functions are executed (lines 2-16). Otherwise, the algorithm terminates by classifying the accelerator as fault-free (lines 17-18).

An accelerator diagnosed as "faulty" is evaluated by $K-1$ fault type specific test images. For a fault type f, if the classification results of t_{high}% or more of its fault specific test images change

than the other faults, f is considered one of the accelerator's major fault type. On the other hand, if this condition does not hold for any of the $K-1$ fault types, the accelerator's fault type is predicted as random (lines 4-9). Meanwhile, an accelerator diagnosed as "faulty" is also evaluated with the selected fault rate sensitive test images. For each image, the ranking of its top-1 label in the original network is collected. For a set with m images, this gives a total number of m ranking values. The severe score is defined as the sum of the average μ_m and standard deviation σ_m of these m values:

$$Severe\ Score = \mu_m + \sigma_m \tag{6}$$

A higher severe score indicates that the change in output class ranking is both more significant and more diverse, indicating higher fault rates which probably demand more costly recovery process. The severe score is compared with two thresholds s_{high} and s_{low} to classify the accelerator into low, medium, and high levels of fault severeness (lines 10-16).

4 EXPERIMENTAL EVALUATION

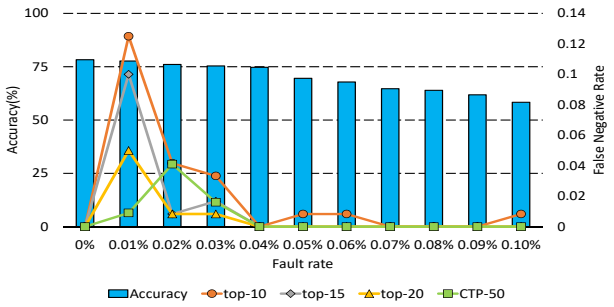
4.1 Experiment Setup

The proposed self-test framework was implemented in Keras [4] and evaluated for CIFAR-10 [11] dataset which contains 50,000 training and 10,000 testing images in 10 classes. To show the general applicability of the proposed framework, we trained two different models, Cifar-Net and VGG-16 on the CIFAR-10 dataset, which achieve a top-1 accuracy of 78.28% and 99.32%, respectively. Since most DNN accelerators utilize quantized weight values to eliminate expensive floating-point arithmetic operations and reduce the model size, we also quantized weight values to four different levels (8–16 bits) for both CIFAR-Net and VGG-16. Each of the quantized models were evaluated under three potential fault types:

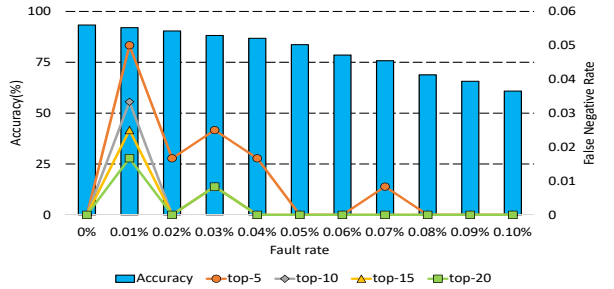
Random: This models the impact of transient and persistent faults on NN accelerators by flipping bits at randomly selected positions. Each bit of each weight is given a predefined rate p to be flipped.
Worst-case: This is designed to simulate fault injection attacks [17, 24] where the attacker intentionally causes the biggest damage. It randomly selects weights according to a predefined rate p_{msb}, and flips the most significant bit (MSB) of the selected weights.
Imprecise Programming & Drifting: This model simulates imprecise programming and drifting of the weight values over time, which are common errors in many emerging devices such as PCM and RRAM [9, 19]. The drifted weight value w_d is modeled with a parameter β which follows a Gaussian distribution identified by standard deviation σ, as shown in Eq. (7). Imprecise programming can be modeled with a zero μ and non-zero σ.

$$w_d = w(1+\beta), \quad \beta \sim \mathcal{N}(\mu, \sigma^2) \tag{7}$$

We performed two sets of fault injections. First, one set of synthetic fault maps were generated for the purpose of test image selection. Specifically, for random, worst-case, and drifting faults, all the quantized network models were evaluated under 7 fault rates with $p/p_{msb}/\mu = \{0.05, 0.1, 0.2, 0.5, 1, 2, 5\}\%$. For each quantization and fault rate combination, 20 different fault maps were randomly generated to mitigate the influence of random variation. Then, another set of synthetic fault maps were generated for evaluating the quality of the selected test images. These fault maps followed a more diverse set of fault rates, ranging from 0.01% to 5%. For each quantization and fault rate combination, 5 different fault maps were randomly generated. Overall, a total number of $3\times4\times7\times20 = 1680$ fault maps were generated for selecting the three sets of test images, while $3\times4\times9\times5 = 540$ fault maps were generated for evaluating them.



(a) Cifar-Net



(b) VGG-16

Figure 4: Fault detection accuracy (measured with false negative rate) vs model accuracy with different fault rates.

4.2 Experimental Results

Fault detection: For both Cifar-Net and VGG-16, images with top 5–20 GS scores were selected for fault detection based on Eqs. (2) and (3). They were evaluated with fault maps of 0.01% to 0.1% fault rates, as it is more challenging to detect models with small fault rates. Fig. 4 shows the accuracy trend and the false negative rates[3] for different sizes of the test set (5–20 images). A larger test set is expected to improve test accuracy (by reducing fault negatives) while increasing test overhead. As Fig. 4(a) shows, when the fault rate is larger than 0.02%, the proposed general test set is more accurate than the CPT-50 set proposed in [16][4]. When the fault rate is 0.01%, the test set needs to include 20 images to achieve low fault negative rate. This, however, does not mean that the proposed test images are insensitive to accuracy drop, as the inference accuracy only drops negligibly at that rate. The results of VGG-16 in 4(b) show similar trends: by enlarging the test set, the false negative rate decreases from 0.05 to 0.02. Overall, a test set of 15 images is a good option for balancing false negative rate and test cost, and its size is only 30% of the CTP-50 set [16].

[3] A false negative occurs if none of the selected test images is misclassified.
[4] We compared to CPT-50 for Cifar-Net but not VGG-16 as only the results of Cifar-Net were reported in [16].

Table 1: Confusion matrix of Cifar-Net fault diagnosis

Real/Predicted	Random	Drifting	Worstcase
Random	**0.38**	0.46	0.15
Drifting	0	**1**	0
Worst-case	0.44	0.11	**0.45**

Table 2: Confusion matrix of VGG-16 fault diagnosis

Real/Predicted	Random	Drifting	Worstcase
Random	**0.6**	0.09	0.31
Drifting	0.18	**0.54**	0.28
Worst-case	0.33	0.04	**0.63**

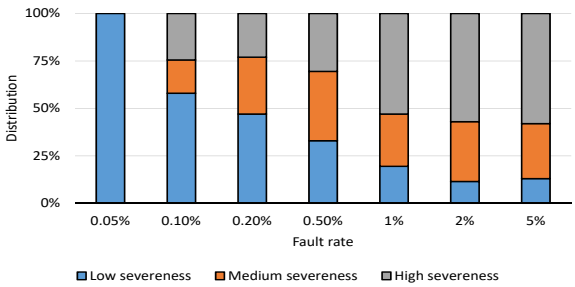


Figure 5: Fault severeness prediction for Cifar-Net

Fault type identification: We selected two different sets of fault type specific images, one for drifting and one for worst-case, each containing 5 images. The corresponding thresholds were set as Th_f = 0.8 and Th_j = 0.6, while the fault rate covered a large range of 0.05% to 5%. We conducted the diagnosis procedure following Algorithm 1. The confusion matrices are shown in Tables 1 and 2. Each row represents a real fault type, while each column represents a predicted fault type. As shown, the diagnosis accuracy of Cifar-Net and VGG-16 are respectively 61% and 59%. Among the three fault types, random and worst-case faults are less distinctive and about 33-40% of fault maps are mis-classified as one another. The drifting test set of Cifar-Net is more sensitive (i.e., few false negatives), while the drifting test set of VGG-16 is more specific (i.e., few false positives). The relatively low accuracy is due to the randomness in generating the fault maps.

Fault severeness prediction: 5 images were selected based on Eq. (5) to monitor the fault severeness of the accelerator, following the method described in Algorithm 1. The prediction results are shown in Fig. 5. When the fault rate is 0.05%, all the fault maps are identified as low severeness. As the fault rate increases to 0.1%, some fault maps are identified as medium/high severeness. When the fault rate increases beyond 1%, the inference accuracy drops to 10% which is equal to random guess accuracy, and the scores of different output classes also become closer. As a result, the majority of fault maps are identified as higher or medium severeness.

5 CONCLUSIONS

In this paper, we proposed a comprehensive self-test framework integrating fault detection, fault type identification, and fault severeness prediction for NN accelerators at run-time. We proposed three algorithms to down select the original test datasets into three small yet effective test image sets. We evaluated the effectiveness of our selected test images and the proposed self-test framework on CIFAR-10 dataset and two network topologies. With only 30 images, the framework achieves high accuracy in fault detection and acceptable accuracy in fault diagnosis and severeness prediction. Outcome of the self-test framework can be used to guide the recovery process of a faulty accelerator.

6 ACKNOWLEDGMENTS

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

REFERENCES

[1] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov. 2012. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* 23, 7 (2012), 075201.
[2] C. Chen, H. Shih, et al. 2015. RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Trans. Comput.* 64, 1 (2015), 180–190.
[3] L. Chen, J. Li, et al. 2017. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *Design Autom. & Test in Europe (DATE)*. 19–24.
[4] F. Chollet et al. 2015. *Keras.* https://github.com/fchollet/keras
[5] C. Constantinescu. 2003. Trends and challenges in VLSI circuit reliability. *IEE Micro* 23, 4 (2003), 14–19.
[6] B. Feinberg, S. Wang, and E. Ipek. 2018. Making memristive neural network accelerators reliable. *Intl. Symp. High Perform. Comput. Archit. (HPCA)* (2018), 52–65.
[7] Z. He, A. S. Rakin, et al. 2020. Defending and harnessing the bit-Flip based adversarial weight attack. In *Comput. vision and pattern recognition (CVPR)*.
[8] S. Hong, P. Frigo, et al. 2019. Terminal brain damage: exposing the graceless degradation in deep neural networks under hardware fault attacks. In *USENIX Security Symp. (USENIX)*. 497–514.
[9] D. Ielmini, A. L. Lacaita, and D. Mantegazza. 2007. Recovery and drift dynamics of resistance and threshold voltages in phase-change memories. *IEEE Trans. Electron Devices* 54, 2 (2007), 308–315.
[10] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu. 2015. Modeling, detection, and diagnosis of faults in multilevel memristor memories. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 34, 5 (2015), 822–834.
[11] A. Krizhevsky, V. Nair, and G. Hinton. 2010. CIFAR-10 (Canadian Institute for Advanced Research). (2010). http://www.cs.toronto.edu/~kriz/cifar.html
[12] B. Li, Y. Wang, Y. Chen, H. H. Li, and H. Yang. 2014. ICE: Inline calibration for memristor crossbar-based computing engine. In *Design Autom. & Test in Europe (DATE)*. 1–4.
[13] S. Li, D. Niu, et al. 2017. DRISA: A DRAM-based reconfigurable in-situ accelerator. In *Intl. Symp. Microarchitecture (MICRO)*. 288–301.
[14] W. Li, W. Wang, H. Li, and X. Li. 2019. RRAMedy: Protecting ReRAM-based neural network from permanent and soft faults during its lifetime. In *Intl. Conf. Comput.-Aided design (ICCAD)*. 91–99.
[15] C. Liu, M. Hu, J. P. Strachan, and H. Li. 2017. Rescuing memristor-based neuromorphic design with high defects. In *Design Autom. Conf. (DAC)*. 1–6.
[16] Qi. Liu, W. Wen, and C. Yang. 2019. Monitoring the health of emerging neural network accelerators with cost-effective concurrent test. In *Design Autom. Conf. (DAC)*. 91–99.
[17] Y. Liu, L. Wei, B. Luo, and Q. Xu. 2017. Fault injection attack on deep neural network. In *Intl. Conf. Comput.-Aided design (ICCAD)*. 131–138.
[18] A. Shafiee, A. Nag, et al. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *Comput. Archit. News* 44, 3 (2016), 14–26.
[19] A. M.S. Tosson, M. Anis, and L. Wei. 2016. RRAM refresh circuit: A proposed solution to resolve the soft-error failures for HfO2/Hf 1T1R RRAM memory cell. In *Great Lakes Symp. on VLSI (GLSVLSI)*. 227–232.
[20] A. J. Van de Goor and Y. Zorian. 1994. Effective march algorithms for testing single-order addressed memories. *Journal of Electronic Testing* 5, 4 (1994), 337–345.
[21] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang. 2017. Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems. In *Design Autom. Conf. (DAC)*. 1–6.
[22] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie. 2015. Overcoming the challenges of crossbar resistive memory architectures. In *Intl. Symp. High Perform. Comput. Archit. (HPCA)*. 476–488.
[23] C. Zhang, P. Li, et al. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Intl. Symp. FPGAs (FPGA)*. 161–170.
[24] P. Zhao, S Wang, et al. 2019. Fault sneaking attack: A stealthy framework for misleading deep neural networks. In *Design Autom. Conf. (DAC)*. 1–6.