

AI Eye Charts: measuring the visual acuity of Neural Networks with test images

*Original*

AI Eye Charts: measuring the visual acuity of Neural Networks with test images / Porsia, Antonio; Ruospo, Annachiara; Sanchez, Ernesto. - ELETTRONICO. - (In corso di stampa). (Intervento presentato al convegno IEEE 2nd International conference on Design, Test & Technology of Integrated Systems tenutosi a Aix-en-Provence (FR) nel 14-16 October 2024).

*Availability:*

This version is available at: 11583/2993100 since: 2024-10-06T16:04:06Z

*Publisher:*

IEEE

*Published*

DOI:

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*

IEEE postprint/Author's Accepted Manuscript

©9999 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

# AI Eye Charts: measuring the visual acuity of Neural Networks with test images

Antonio Porsia  
Politecnico di Torino  
DAUIN  
Turin, Italy  
antonio.porsia@polito.it

Annachiara Ruospo  
Politecnico di Torino  
DAUIN  
Turin, Italy  
annachiara.ruospo@polito.it

Ernesto Sanchez  
Politecnico di Torino  
DAUIN  
Turin, Italy  
ernesto.sanchez@polito.it

**Abstract**—In the last decade, neural networks (NNs) have established themselves as the foundation of modern computer vision, achieving remarkable performances in a consistent number of tasks, including safety-critical applications such as autonomous driving and healthcare devices. As a consequence, existing devices, such as Graphics Processing Units (GPUs), have evolved and new, specialized AI hardware has been designed to better support the ever-growing demand for computational resources. Since the adoption of NNs in safety-critical tasks requires to ensure that hardware faults are not negatively affecting the application's performance, the use of self-test routines to continuously verify the device's functionality during operation is crucial. However, conventional self-testing methods such as Built-in Self Test (BIST) and Software Test Libraries (STLs) are not ideal candidates for this task, since they require to completely halt the application's operation at test time. For this reason, various methods have been proposed that employ the NN itself to test the hardware it is running on by launching the inference of one or more test images. The aim of this paper is to examine the current state-of-the-art regarding test images, as well as draw a comparison of the methods that have been proposed to choose and/or generate test images.

## I. INTRODUCTION

The ever-growing use of Neural Networks (NNs) in computer vision tasks, such as image classification, semantic segmentation and object detection, has raised questions over their reliability when used in safety-critical applications, such as autonomous driving and healthcare devices. Requirements for functional safety force manufacturers and users to implement safety mechanisms that are able to continuously assess and ensure the correct functioning of the application during its in-field use. Conventional hardware self-test mechanisms such as Built-In Self-Test (BIST) and Software Test Libraries (STLs) may be used to ensure the application's correctness at use time.

However, the hardware overhead of BIST and the requirement to halt the application's operation at test time of both BIST and STL methods may hinder the performance of NNs [1]. For this reason, the academic community has strived to look for alternative strategies that are able to achieve a good compromise between testing capabilities and performance.

In particular, using a set of input images as a test vector seems to be a promising technique to assess the correct functionality of a neural network, since neural network inference is a fast process

and does not require halting the network's operation. The first occurrence of test images in the literature seems to be in [2], where the authors develop a set of images to assess the correct functionality of a Deep Neural Network (DNN)-based autonomous driving system. However, this work focuses on behavioral testing, i.e., it does not aim to detect erroneous behavior due to hardware, but due to the network itself. Nonetheless, it served as a starting point to extend the concept of test images to hardware testing.

The first work explicitly targeting hardware faults occurring in an accelerator used for the execution of a Neural Network is [3]. The authors generate adversarial examples with the explicit aim of detecting hardware faults occurring in a Resistive RAM (ReRAM)-based hardware accelerator for the execution of a neural network. Since then, a plethora of different methods of generating test image sets has appeared in the literature.

This work aims to offer a comprehensive description of the state of the art regarding test images, as well as draw a comparison of the available methods from various viewing angles.

The paper is organized as follows: section II provides a description and classification of the various test image construction methods and section III draws conclusions.

## II. TEST IMAGE TAXONOMY

The test images found in the literature can be classified according to several criteria:

- **Test image construction method** – test images can be developed through various techniques:
  - *Selection* – already existing images are selected according to a certain metric
  - *Generation* – new images are specifically constructed to have desirable qualities
  - *Mutation* – existing images are selected and then altered to enhance them with desirable qualities. It can be viewed as a mix of selection and generation
- **Testing methodology** – test images can be developed to either perform:
  - *Behavioral testing* – Test images developed for behavioral testing target the NN itself, rather than the hardware it is running on: network parameters, inputs and activations become potential fault sites
  - *Functional testing* – Test images developed for functional testing use the NN as a means of delivering functional test patterns to the targeted device, using the

TABLE I: Test image construction methods

Reference	Method	Testing methodology	Specificity	Observation point	Target architecture	Approach
[4]	Selective	Behavioral/Functional	Neuromorphic devices	Prediction	SNN	Black-box
[5]	Selective	Behavioral	Hardware-agnostic	Prediction	SNN	Black-box
[6]	Selective	Behavioral/Functional	Memristive devices	Prediction	Model-agnostic	White-box
[7]	Selective	Behavioral	Hardware-agnostic	Prediction	Model-agnostic	Black-box
[8]	Generative	Functional	GPU multipliers	First CONV layer output/confidence scores	CNN	White-box
[9]	Generative	Behavioral/Functional	Memristive devices	Confidence scores	CNN	Black-box
[10]	Generative	Behavioral/Functional	Systolic arrays	Prediction	CNN	Black-box
[11]	Generative	Behavioral	Hardware-agnostic	Prediction	CNN	Black-box
[12]	Generative	Behavioral	Hardware-agnostic	Confidence scores	CNN	Black-box
[3]	Mutation-based	Behavioral/Functional	Memristive devices	Prediction/Confidence scores	Model-agnostic	Black-box
[13]	Mutation-based	Behavioral/Functional	Memristive devices	Prediction	CNN	White-box
[14]	Selective/Generative	Behavioral	Hardware-agnostic	Prediction	CNN	White-box
[15]	Selective/Generative	Behavioral/Functional	Memristive devices	Prediction/Confidence scores	CNN	Black-box

network as a proxy to assess the correct functionality of the underlying hardware

- **Hardware specificity** – Test images can target a specific device or be hardware agnostic. Note that only test images for behavioral/functional testing may be hardware agnostic, since structural testing is by definition hardware-specific
- **Observation point** – Faults are detected by comparing results produced during the inference of the test images against the expected ones. These results may be collected at various points during the inference: they can be intermediate results such as the output feature map of a specific layer, confidence scores or prediction labels
- **Target network architecture** – Test images usually target a specific neural network architecture, with the obvious candidates being the most popular architectures used in computer vision tasks, such as Convolutional Neural Networks (CNNs) and Spiking Neural Networks (SNNs)
- **Black-box or white-box approach** – Test image construction methods may require knowledge of the internal structure of the model or not

Table I summarizes these characteristics for all the research works reported in this section.

#### A. Selective methods

1) *El-Sayed et al.*: El-Sayed et al. [4] proposed a method to select test images for both behavioral and structural testing of neuromorphic circuits running Spiking Neural Networks. Samples  $t_i$  from the training and test sets are ranked according to a *quality metric*  $q_i$  that measures the fault detection capability of an image and is defined as

$$q_i = \frac{1}{n_i^{(1)} - n_i^{(2)}}$$

where  $n_i^{(1)}$  and  $n_i^{(2)}$  are the spike counts of the output neurons associated to the top-1 and top-2 classes, respectively. It follows that samples that are difficult for the SNN to classify, i.e., whose top-1 and top-2 spike counts are very close, are ranked higher. The underlying hypothesis is that samples that are difficult to classify are more sensible to hardware faults and have a higher probability of being misclassified in presence of a fault. The authors performed behavioral testing on N-MNIST and IBM DVS gesture

SNNs using SNN-specific fault models, such as *dead/saturated neurons* and *timing variations* for neurons, and a hardware-aware fault model for synapses that consists in quantizing the real-valued weight, performing a bit-flip and dequantizing the altered weight to a real number. Structural testing has been performed on a custom neuromorphic circuit that implements a SNN for poker card symbol classification. The SNN parameters, represented on 8-bit integers, are stored in on-die memories. The fault model consists in single bit-flips for all parameters and all bit positions and multiple bit-flips with different Bit Error Rate probabilities. This method is shown to detect 100% of critical faults, i.e., faults that alter the network’s response above a certain margin of tolerance, with a test set of around 30 images.

2) *Hsieh et al.*: Hsieh et al. [5] proposed a method to perform hardware-agnostic behavioral testing of SNNs that consists in:

- 1) Measuring the distribution of the  $F_1$  score of a fault-free SNN model by performing Monte Carlo simulations
- 2) Measuring the  $F_1$  score of a faulty SNN model and checking if it is within an acceptable margin from the fault-free average  $F_1$

The authors used behavioral-level fault models for SNNs, targeting neurons and synapses. One noteworthy characteristic of this method is that it uses the *entire dataset* as a test vector. The obtained experimental results show that the  $F_1$  score significantly degrades in presence of specific types of *connectivity faults*, namely Wrong Connection Faults, where spikes are sent to the wrong neurons, and Disconnected Faults, where spikes cannot be sent to a neuron. The failure rate obtained by fault simulation for these two fault models is 89.84% and 90.25%, respectively, while other fault models (spike, weight and delay faults) have a failure rate of 10% or less.

3) *Ahmed and Tahoori*: Ahmed and Tahoori [6] proposed a method to test DNNs implemented on memristive devices by selecting appropriate images from the training set. At each training step, the difference  $\Delta\mathcal{L}$  of the loss function before and after the parameter update is computed for each training sample, then it is accumulated and stored on a per-sample basis. After training, samples are ranked according to the accumulated loss variation and the ones with the highest  $\Delta\mathcal{L}$  are added to the test image set. The underlying hypothesis is that parameters that require more parameter tuning during training will be more sensitive to parameter perturbations.

The proposed test flow consists in launching the inference of samples in the test set and checking if the variation in accuracy  $\Delta A_{\text{infer}}$  with respect to the fault-free NN is over a certain threshold. The authors report that this method is effective when the threshold is relatively high, e.g.  $\Delta A_{\text{infer}} > 2\%$ , while lower thresholds require a different method. The latter consists in storing the average loss  $\mathcal{L}_\tau$  of the test image set and comparing it to the average loss at test time  $\mathcal{L}_P$ . For what concerns the faults model, the authors model conductance variations affecting memristors as additive and multiplicative Gaussian noise applied to the weights, while stuck-at-0/1, stuck-open/short, read/write disturbances and slow-write faults are modeled as a random percentage of weight bits set to either 0 or 1.

The authors report their method reaches 100% fault coverage on three different NN models: a Multi-Layer Perceptron (MLP) trained on the Fashion-MNIST dataset, a ResNet-18 CNN trained on CIFAR-10 both with and without data augmentation and a ResNet-110 trained on CIFAR-100. 100% conductance variation coverage is reached with a minimum number of 16 test images, while 100% coverage of permanent faults is reached with a minimum of 64 test images.

4) *Meng et al.*: Meng et al. [7] proposed a method to select test images from the test set according to their *sensitivity*, defined as:

$$(S_f)_I = \frac{N_{\text{dev}}}{N_{\text{maps}}}$$

where  $N_{\text{maps}}$  is the number of fault maps randomly generated with different fault models and fault rates and  $N_{\text{dev}}$  is the number of times that the prediction for image  $I$  changes with respect to the fault-free one. A *General Score* is then defined to measure the sensitivity of an image  $I$  to various fault types. If there are  $K$  fault types, the General Score is defined as:

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

where  $\sigma_I$  is the standard deviation of the sensitivity of image  $I$  to the various fault types. Images are then ranked by their General Score and the ones with the highest score are selected.

The authors experimentally validated their method by applying it to the Cifar-Net and VGG-16 CNNs trained on the CIFAR-10 dataset and quantized to 8/16-bit integers. The authors employed random bit-flips to model transient faults in the network weights and fault injection attacks and added Gaussian noise to the weights to model imprecise programming and value drifting in novel devices like Resistive RAMs. Five to twenty test images are selected from the test set and then fed to a faulty network with a fault rate ranging from 0.01% to 1%. The configuration with 20 test images is shown to achieve an extremely low false negative rate even with a 0.01% fault rate.

## B. Generative methods

1) *Ruospo et al.*: Ruospo et al. [8] proposed a method to perform on-line self-test of functional units in a GPU running a CNN by using test images. Their method consists in generating test patterns for GPU multipliers by using the weights of the first convolutional layer of the CNN as constraints: the ATPG process

produces an input value corresponding to the input-weight pair, then the input part is split from the weight and placed in a test image by exploiting knowledge of the dataflow algorithm, i.e., the way in which the convolution workload is distributed among the GPU cores. This set of test images is defined Image Test Library (ITL). When the ITL is passed to the CNN, the multipliers receive as inputs the correct input-weight pairs corresponding to test patterns. The output feature map of the first convolutional layer is then compared against the correct one.

Experimental results of gate-level fault simulations show that a 6-image ITL built for ResNet-20 and a 8-image ITL built for DenseNet-121, both trained on the CIFAR-10 dataset, are able to reach 94.74% and 95.46% coverage of stuck-at faults on the multipliers of a GPU, with a self-test time of under a millisecond in both cases.

Recent work [16] reported promising results about ITLs being able to propagate faults up to the output confidence scores. The propagation analysis is performed by performing a gate-level fault simulation for each convolutional layer of the CNN. The multiplications performed during the convolution are subjected to a fault injection to gather faulty outputs, which are then used to reconstruct the faulty output feature map of the layer. The process is then repeated until the last convolutional layer.

2) *Ahmed and Tahoori*: Another work by Ahmed and Tahoori [9] proposes a method to test a Deep Neural Network running on a memristive device with a single image by observing the Kullback-Leibler (KL) divergence between the distribution of the confidence scores of the fault-free and faulty model. The test image is constructed by applying gradient descent to an input image with the learning objective of having the output distribution of the image match a standard Gaussian distribution with zero mean and unit standard deviation.

The authors model conductive variations in memristive technologies as additive and multiplicative Gaussian noise, while random bit-flips with varying fault rates are used to model stuck-at faults, retention faults, read/write disturbances and slow-write faults. The authors validated their method on 6 CNN models used for image classification trained on the ImageNet dataset and 2 CNN models used for semantic segmentation, trained on the Brain MRI and COCO datasets. For each model, a Monte Carlo simulation consisting of 1,000 different device instances has been performed to evaluate the fault coverage reached by this method. The fault coverage is defined by the authors as the ratio between the number of simulation runs in which the KL divergence exceeded a certain threshold  $t$  and the total number of runs. Results show that this one-shot testing method rarely goes below 99% fault coverage, for various noise scales, fault rates and thresholds.

3) *Chaudhuri et al.*: Chaudhuri et al. [10] use Bayesian optimization to generate test images targeting a systolic array-based accelerator. More specifically, the Bayesian optimization engine samples an image from the input image space and tries to maximize the cross-entropy loss between the output of the fault-free accelerator and the output of the faulty accelerator. To avoid generating a separate input image for each fault in the fault list, the average of the losses produced by faulty instances of the accelerator, each presenting a separate fault, is used as the objective function. After  $k$  optimization steps, where  $k$  is a user-

defined parameter, the current image is saved and detected faults are dropped from the fault list. The procedure then restarts to generate a test image for the remaining faults.

To validate their method, the authors inject stuck-at faults in partial-sum and multiplier-output buses of 10 random-chosen PEs of the systolic array. Test images generated with this method for a LeNet-5 CNN trained on the MNIST dataset reach a 90% coverage of faults in sign and exponent bits of 32-bit floating-point values with 10-11 test images, 87% coverage of faults in sign and exponent bits of 16-bit floating-point values with 7-9 test images and 30% coverage of faults in mantissa bits of 32-bit floating-point values.

4) *Kundu and Basu*: Kundu and Basu [11] proposed using a conditional Generative Adversarial Network (cGAN) to generate test images for a target DNN. To evaluate their method, the authors inject permanent bit-flips in the Most Significant Bit of randomly selected 8-bit integer weights of a LeNet-5 CNN trained on the MNIST dataset. A fault is considered detected if the input image is misclassified with respect to the fault-free model. Results show that 30 cGAN-generated test images cover 40% of faults with a fault rate of 0.025% and reach 100% coverage with a 0.4 % fault rate.

5) *Turco et al.*: Turco et al. [12] proposed a method that leverages an evolutionary algorithm to generate an Image Test Library to detect permanent, hard-to-detect faults and observe their effects on the network's output. The fitness function of the evolutionary algorithm is defined as the ratio between the number of faults whose effect propagates to the network's output and the total number of injected faults.

To experimentally validate their method, the authors inject bit-flips in the sign and mantissa bits of 32-bit floating-point weights of three different CNNs trained on CIFAR-10: ResNet-18, ResNet-34 and DenseNet-161. Sign and mantissa bits were explicitly singled out due to faults occurring in those positions rarely affecting the network's output. Two experiments were carried out, one targeting the whole output vector of the network (Logit) and one targeting only the maximum value of the output (Max-Logit), i.e., the one corresponding to the prediction. Results show that evolutionary-based 50-image ITLs are able to detect over 75% of injected faults.

### C. Mutation-based methods

1) *Li et al.*: Li et al. [3] proposed to use adversarial examples to test ReRAM-based accelerators running a neural network. They hypothesize that since adversarial examples are generated according to the loss gradient (when using the Fast Gradient Sign Method), they are particularly sensible to weight variations.

The authors targeted both permanent faults, such as stuck-at-0/1, and soft faults affecting ReRAM-based devices. To experimentally validate their method, the authors employed three different NNs, namely a 3-layer MLP trained on the MNIST dataset and two CNNs, LeNet-5 trained on MNIST and ConvNet-quick trained on CIFAR-10. Then they constructed 100 faulty models for each architecture by randomly injecting faults, with fault rates of 0.05% (MLP and LeNet-5) and 0.02% (ConvNet) for permanent faults and 1% (MLP and LeNet-5) and 0.04% (ConvNet) for soft faults. The authors used SDC-1 and SDC-3% as detection metrics

and evaluated the method performance by defining a Detection Accuracy metric, defined as:

$$DA = \frac{\text{N. of detected faulty models}}{\text{N. of faulty models}}$$

Using SDC-3% as a detection metric, the proposed method achieves a 99% detection accuracy on the MLP and ConvNet and 100% on LeNet-5.

2) *Chen et al.*: Chen et al. [13] developed a set of test images by introducing a backdoor into a DNN. Their approach consists in watermarking images from the validation set and fine-tuning the network on the watermarked dataset, introducing a backdoor into the DNN that acts as a "checksum" function. The authors managed to make the watermarked images extremely sensitive to weight deviations by purposely overfitting the model when training on watermarked images.

This method explicitly targets ReRAM-based accelerators, modeling faults as random bit-flips that occur with a certain probability. The authors use a custom metric for evaluation called Accuracy Deviation (AD), defined as:

$$AD = \frac{\text{Fault-free accuracy} - \text{Deviation-affected accuracy}}{\text{Fault-free accuracy}}$$

Experimental validation has been performed targeting the AlexNet and VGG-16 CNNs trained on the CIFAR-10 dataset and pruned to remove 90% of parameters without accuracy loss. The entire validation set (10,000 images) is then watermarked and used to fine-tune the model to insert the backdoor. The authors report that their method achieves a mean AD of 0.72 (AlexNet) and 0.89 (VGG-16) for stuck-at-0/1 faults and 0.78 (AlexNet) and 0.89 (VGG-16) for stuck-open/short faults, with fault rates varying from 0.1% to 2%.

### D. Mixed methods

1) *Luo et al.*: Luo et al. [14] proposed a behavioral testing method for Deep Neural Networks (DNNs) that consists in selecting images from the training set that are able to activate as much model parameters as possible. A test image is said to activate a parameter when perturbations of the parameter are propagated to the output during the inference of said image. The authors determine whether a parameter is activated by a test image by observing if the gradient of the DNN with respect to the parameter is non-zero. The *validation coverage* of a test image  $\mathbf{x}$  is defined as:

$$VC(\mathbf{x}) = \frac{\#\{\theta \mid \nabla_{\theta} F(\mathbf{x}) \neq 0\}}{\#(\theta)}$$

where  $F(\cdot)$  is the function that describes the DNN and  $\theta$  is a DNN parameter. The test image set is constructed by selecting images in an iterative manner, where at each step the image with the highest validation coverage is added to the set, until the set reaches a certain size.

To increase the overall validation coverage, the authors propose to generate *synthetic samples* using the gradient descent method that are able to activate the remaining parameters. The original model is stripped of the parameters already activated by test images extracted from the training set, and a batch of  $k$  synthetic

samples, where  $k$  is the number of output neurons, is updated using gradient descent until the remaining parameters are activated.

The performance of the proposed scheme is validated on two custom CNNs trained respectively on the MNIST and CIFAR-10 datasets. Test image sets constructed with the above method reach a detection rate of 96.1% and 95.2% when random gaussian noise is added to the model parameters.

2) *Liu et al.*: Liu et al. [15] proposed a method to test ReRAM (Resistive RAM)-based DNN accelerators by selecting from the training dataset images that are as equidistant as possible from all decision boundaries, i.e., images whose confidence scores produced by the DNN are very close to each other. Since it is difficult to find images whose confidence scores are perfectly equal, the authors propose to generate  $k$  other images, where  $k$  is the number of output classes, to use in conjunction with the dataset ones. The generated images are optimized by minimizing a cross-entropy loss function with two targets:

- 1) Generating an image that makes the fault-free network output an almost equal probability for all classes
- 2) Generating an image that makes a faulty network output an extremely high probability for a certain class

The relative importance of the two terms in the loss function is controlled by a parameter  $0 < \alpha < 1$ .

The authors evaluated their method by modeling the impact of weight drifting faults and random soft errors on two CNNs: LeNet-5 trained on the MNIST dataset and ConvNet-7 trained on CIFAR-10. The authors selected 50 images from the dataset using the first method and generated 10 other images using the second method, for a total test image set size of 60 images. Metrics used in the evaluation are:

- 1) SDC-1/5 – difference between top-1/5 classes between the fault-free and faulty models
- 2) SDC-T5%/T10% – difference in percentage between top ranked confidence scores produced by the fault-free and faulty models
- 3) SDC-A3%/A5% – difference between the average of confidence scores produced on the generated images by the fault-free and faulty models

The authors report that their method reaches a detection rate higher than 94% across all metrics, except for SDC-1.

### III. CONCLUSIONS

The increasing use of Artificial Neural Networks in safety-critical computer vision tasks mandates for in-field testing methods that are less invasive and disruptive as possible.

This paper tried to provide a comprehensive description of the state-of-the-art regarding test images for neural network testing. As it has been shown, the available literature provides several alternatives that cover a variety of specific needs that may arise in real-world applications, suggesting that test images are becoming a viable alternative to classical hardware self-test routines.

### REFERENCES

- [1] A. Ruospo, D. Piumatti, A. Floridia, and E. Sanchez, "A suitability analysis of software based testing strategies for the on-line testing of artificial neural networks applications in embedded devices," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2021, pp. 1–6.
- [2] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 303–314.
- [3] W. Li, Y. Wang, H. Li, and X. Li, "RRAMedy: Protecting ReRAM-Based Neural Network from Permanent and Soft Faults During Its Lifetime," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, Nov. 2019, pp. 91–99, iSSN: 2576-6996.
- [4] S. A. El-Sayed, T. Spyrou, L. A. Camuñas-Mesa, and H.-G. Stratigopoulos, "Compact Functional Testing for Neuromorphic Computing Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2391–2403, Jul. 2023, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.
- [5] Y.-Z. Hsieh, H.-Y. Tseng, I.-W. Chiu, and J. C. M. Li, "Fault Modeling and Testing of Spiking Neural Network Chips," in *2021 IEEE International Test Conference in Asia (ITC-Asia)*, Aug. 2021, pp. 1–6, iSSN: 2768-069X.
- [6] S. T. Ahmed and M. B. Tahoori, "Compact Functional Test Generation for Memristive Deep Learning Implementations using Approximate Gradient Ranking," in *2022 IEEE International Test Conference (ITC)*, Sep. 2022, pp. 239–248, iSSN: 2378-2250.
- [7] F. Meng, F. S. Hosseini, and C. Yang, "A Self-Test Framework for Detecting Fault-induced Accuracy Drop in Neural Network Accelerators," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. New York, NY, USA: Association for Computing Machinery, Jan. 2021, pp. 722–727.
- [8] A. Ruospo, G. Gavarini, A. Porsia, M. S. Reorda, E. Sanchez, R. Mariani, J. Aribido, and J. Athavale, "Image Test Libraries for the on-line self-test of functional units in GPUs running CNNs," in *2023 IEEE European Test Symposium (ETS)*, May 2023, pp. 1–6, iSSN: 1558-1780.
- [9] S. T. Ahmed and M. B. Tahoori, "One-Shot Online Testing of Deep Neural Networks Based On Distribution Shift Detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.
- [10] A. Chaudhuri, C.-Y. Chen, J. Talukdar, and K. Chakrabarty, "Functional Test Generation for AI Accelerators using Bayesian Optimization," in *2023 IEEE 41st VLSI Test Symposium (VTS)*, Apr. 2023, pp. 1–6, iSSN: 2375-1053.
- [11] S. Kundu and K. Basu, "Detecting Functional Safety Violations in Online AI Accelerators," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, Sep. 2022, pp. 1–4, iSSN: 1942-9401.
- [12] V. Turco, A. Ruospo, G. Gavarini, E. Sanchez, and M. S. Reorda, "Uncovering hidden vulnerabilities in cnns through evolutionary-based image test libraries," in *2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2023, pp. 1–6.
- [13] C.-Y. Chen and K. Chakrabarty, "On-line Functional Testing of Memristor-mapped Deep Neural Networks using Backdoored Checksums," in *2021 IEEE International Test Conference (ITC)*, Oct. 2021, pp. 83–92, iSSN: 2378-2250.
- [14] B. Luo, Y. Li, L. Wei, and Q. Xu, "On Functional Test Generation for Deep Neural Network IPs," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2019, pp. 1010–1015, iSSN: 1558-1101.
- [15] Q. Liu, T. Liu, Z. Liu, W. Wen, and C. Yang, "Monitoring the Health of Emerging Neural Network Accelerators with Cost-effective Concurrent Test," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, Jul. 2020, pp. 1–6, iSSN: 0738-100X.
- [16] A. Bosio, M. Gomes, F. Pavanello, A. Porsia, A. Ruospo, E. Sanchez, and E. I. Vatajelu, "Resiliency approaches in convolutional, photonic, and spiking neural networks," in *2024 IEEE 25th Latin American Test Symposium (LATS)*, 2024, pp. 1–10.