

UC Irvine

UC Irvine Previously Published Works

Title

A New Online Fault Detection Mechanism for Neural Network Applications

Permalink

<https://escholarship.org/uc/item/6qg5w3z2>

Journal

2025 32nd IEEE International Conference on Electronics, Circuits and Systems (ICECS)

Authors

Paola, Luca De

Esposito, Giuseppe

Guerrero-Balaguera, Juan-David

et al.

Publication Date

2025-11-17

DOI

10.1109/icecs66544.2025.11270797

Copyright Information

This work is made available under the terms of a Creative Commons Attribution-NoDerivatives License, available at <https://creativecommons.org/licenses/by-nd/4.0/>

Peer reviewed

A New Online Fault Detection Mechanism for Neural Network Applications

Luca De Paola, Giuseppe Esposito, Juan-David Guerrero-Balaguera, and Matteo Sonza Reorda

Politecnico di Torino - Department of Control and Computer Engineering (DAUIN)

luca.depaola@studenti.polito.it, {giuseppe.esposito, juan.guerrero, matteo.sonzareorda}@polito.it

Abstract—This paper proposes a novel strategy for detecting hardware-induced misbehaviors during the execution of Convolutional Neural Networks (CNNs). The fault detection strategy relies on an auxiliary lightweight Multi-Layer Perceptron (MLP) acting on the outputs of the first layer of the monitored CNN as an early exit instrument, and enabling the detection of faults possibly affecting the CNN computations. Specifically, the outcomes of the CNN and the MLP are compared across a sequence of observations (observation window), calculating the level of agreement between them. As a result, an incorrect prediction from a CNN caused by a hardware fault can be identified when the agreement drops below a set threshold. This mechanism was implemented and evaluated using a set of representative CNNs and datasets. The results showed that the system can detect between 78.60% to 97.79% of faults, depending on the CNN and the dataset.

Index Terms—Hardware Fault Detection, Convolutional Neural Networks, Reliability, AI Accelerators

I. INTRODUCTION

Artificial Intelligence (AI) systems based on Deep Neural Networks (DNNs) and especially Convolutional Neural Networks (CNNs) are widely spread into multiple application domains, including safety-critical applications, such as autonomous vehicles, healthcare systems, and industrial automation [1]. In these contexts, ensuring the reliability of neural network computations is paramount to guarantee the proper operation of the system even under harsh conditions or unexpected defects affecting the underlying hardware.

Nowadays, the continuous scaling of semiconductor technologies has introduced new reliability challenges. The increasing density and complexity of modern hardware architectures make neural networks highly susceptible to defects, which can manifest as bit-flips in different components of the computational circuit, memories, registers, or the outcomes of arithmetic units. Indeed, recent studies have demonstrated that some faults are especially dangerous, since they silently induce and propagate errors through the system computations, leading to an unnoticed functional misbehavior [2]. The error can propagate to the application, corrupting the CNN intermediate outputs at inference time without causing a system crash (e.g., memory access violation), consequently inducing a change in the CNN outcome. These fault effects can lead to catastrophic outcomes, including the loss of human lives. Therefore, it is

crucial to develop mechanisms at the application level that promptly detect any wrong outcomes during the CNN operation, contributing to the early detection of underlying hardware faults and enabling the execution of specialized hardware test routines or the activation of fault countermeasures effectively.

Several works have proposed different strategies for the detection of CNN misbehaviors due to hardware faults. Some of these works are based on in-neuron anomaly detection [3], [4], checksums [5], [6], neuron profiling [7], [8], artificial test data generation [9], [10], and Open-set recognition [11].

While existing literature offers various strategies for detecting CNN malfunctions, they focus on individual inference samples. Nonetheless, in real deployments, CNNs process sequential input observations (i.e., images), often producing repeated outcomes over time. This work introduces a fault detection mechanism that, at each inference, evaluates both current and recent outcomes within a time window.

Specifically, the proposed detection mechanism relies on an auxiliary lightweight Multi-Layer Perceptron (MLP) monitoring the first layer of the main CNN, acting as an early exit instrument [12], and enabling the detection of faults across the CNN computations. The anchor mechanism between the CNN and the MLP corresponds to a feature extractor applied to the feature maps of the first layer of the CNN. This approach exploits the statistical properties of intermediate feature maps, such as mean values, percentiles, and simple spatial descriptors such as the activation centroid. These extracted features are used to train the MLP to replicate the same behavior of the reference CNN model. During the operational phase, both neural networks (the original one and the MLP) operate in parallel, producing inference results that are later used to compute the level of agreement. Such an agreement is calculated by comparing the inference outcomes across the time window of observations. In the end, a fault is detected when the level of agreement between the MLP and the main CNN drops under a threshold previously established using the fault-free validation dataset.

It is worth noting that the proposed method requires no architectural changes and can be deployed on pre-trained models. Moreover, it is class-agnostic, label-free, and able to distinguish faults of varying severity. Our experimental results demonstrate that the proposed system can effectively detect a broad range of injected faults, particularly those that lead to critical Silent Data Corruptions (SDCs), with minimal false positives, across various datasets and architectures. The

Partially supported by the PNRR through the National Center for HPC, Big Data and Quantum Computing. We also acknowledge ISCRA (HardNet) for the access to the LEONARDO supercomputer (EuroHPC JU), hosted by CINECA, Italy.

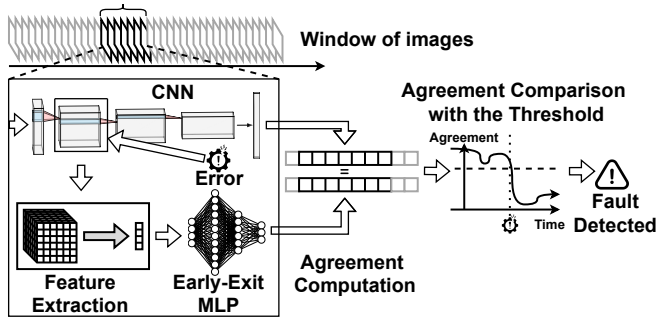


Fig. 1. Overview of the proposed detection process, showing feature extraction, parallel MLP inference, and agreement computation over a time window.

experimental results show that the detection system maintains a false positive rate of no more than 7.33%. Additionally, it can correctly identify faulty cases up to 97.79% of the time.

II. PROPOSED METHOD

The proposed fault detection mechanism considers a sequence of consecutive inference observations to detect the presence of faults affecting a CNN. It leverages basic statistical features extracted from the Output Feature Maps (OFMs) of the first convolutional block of the CNN, which are used to train an early-exit, lightweight MLP, which learns to predict the CNN's final classification outputs. During inference, both the CNN and the MLP produce predictions in parallel. Faults are detected by computing the *agreement* between the CNN and the early-exit classifier over an observation window (i.e., the past I inferences), updating the agreement at each new inference step; a fault is considered detected when the agreement metric falls under a threshold. A visual overview of the overall fault detection process is provided in Figure 1.

Although features are extracted from the first convolutional layer, faults occurring deeper in the CNN can still be detected. This is because the MLP receives clean features, extracted before the fault occurs, and learns to replicate the fault-free CNN's behavior. When a fault in a later layer causes the CNN to misclassify, the MLP still predicts the correct class, leading to a drop in agreement. This divergence allows the detection mechanism to identify faults anywhere in the network.

The fault detection mechanism is implemented via a design flow comprising four stages: *i*) feature extraction, *ii*) dataset generation and early-exit MLP training, *iii*) agreement threshold computation and *iv*) fault detection validation.

A. Feature Extraction

During training inference, OFMs from convolutional and ReLU layers of a selected block are extracted along with the CNN's final prediction. For each OFM, a set of statistical measures is computed both globally and per channel, treating the feature map as a multidimensional array of 2D activation channels. The extracted statistics compactly summarize activation distributions within the OFMs and include: Mean, Standard Deviation, Minimum, Maximum, Median, and Percentiles (25th, 75th). Moreover, the Positive Activation Ratio and Center of Mass (X, Y) are extracted to capture activation sparsity and spatial distribution, respectively [13].

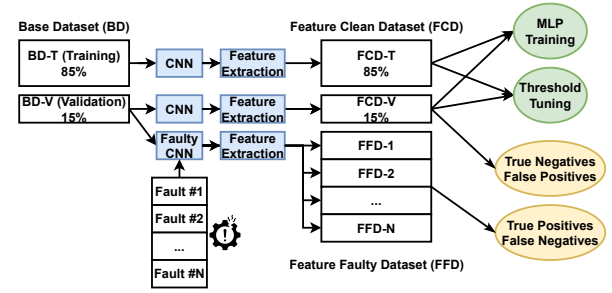


Fig. 2. Dataset generation pipeline for the training and validation steps of the design process of the fault detection mechanism.

After feature extraction, a filtering step is applied to discard invalid or redundant statistics. This step enhances the computational and storage efficiency of the fault detection pipeline and improves the performance of the early-exit MLP by reducing the dimensionality of its input. Invalid features, such as those containing non-computable values (e.g., NaNs), are first removed; these often arise from operations like division on zero-only activations caused by ReLU-induced sparsity. Redundant features are then identified and discarded through pairwise Pearson correlation analysis: when two features exhibit a correlation coefficient above 0.9, one is eliminated, following standard filter-based approaches for feature selection [14]. Finally, features with low variability—those taking the same value in more than 30% of the samples—are excluded, as they are unlikely to be informative for training (e.g., the minimum activation after a ReLU, often equal to zero).

B. Dataset Generation and Early Exit MLP training

Dedicated datasets of statistical features are constructed using the feature extraction pipeline described earlier, as illustrated in Figure 2. Starting from the original image dataset—referred to as the Base Dataset (BD)—a standard holdout split into training (BD-T 85%) and validation (BD-V, 15%) sets [14] is applied. These subsets are processed by the fault-free CNN, and the selected statistical features are extracted for each sample, resulting in the Feature Clean Dataset (FCD), composed of training (FCD-T) and validation (FCD-V) partitions. To ensure reliable modeling, only correctly classified samples from FCD-T are used to train the lightweight MLP, so that it learns to replicate the correct behavior under fault-free conditions.

To evaluate the system under fault conditions, a set of Feature Faulty Datasets (FFDs) is generated. Specifically, N distinct faults are injected—one per model instance—into the CNN and the same BD-V inputs are processed, yielding N corresponding faulty datasets (FFD-1, ..., FFD- N), each containing features extracted from the fault-affected networks.

C. Agreement Threshold Computation

The agreement is formally defined as the proportion of matching predictions between the CNN and the MLP over a fixed number of recent inferences. To capture this, a sliding window of size I , which stores the prediction outcomes of the past I samples, is employed. At each new inference, the window is updated by removing the oldest result and including

the latest one, allowing the agreement score to evolve dynamically over time. This sliding mechanism smooths out short-term fluctuations and enhances the robustness of detection, particularly in the presence of transient misclassifications.

To distinguish between normal disagreement and actual faults, an agreement threshold is defined. This threshold is derived from the FCD-T dataset by applying the same sliding window procedure and recording the agreement values. The minimum agreement value observed across all windows is then selected as the threshold. This conservative approach accounts for natural variability between the two models and minimizes the risk of false positives during inference.

A fault is flagged during normal operation whenever the agreement over the current window drops below the threshold. Although the window size is fixed to $I = 300$ in this work, it can be tuned to balance sensitivity and stability.

D. Fault Detection Validation

The validation process focuses on two key components of the proposed methodology: the early-exit MLP and the agreement threshold. It relies on the datasets introduced in the design phase, namely FCD-T, FCD-V, and FFDs.

The MLP is validated on FCD-V, considering only the samples for which the CNN provides correct predictions. This setup ensures that the secondary classifier is evaluated under the same conditions it was trained to emulate.

The agreement threshold is computed on FCD-T by calculating agreement values over sliding windows. The threshold is set to the minimum observed agreement, ensuring the absence of false positives in fault-free training conditions.

Fault detection performance is evaluated using both FCD-V and FFDs, according to the following metrics: *i)* True Positives (TPs): proportion of injected faults (from FFDs) whose agreement is below the threshold; *ii)* False Positives (FPs): proportion of windows from FCD-V where agreement falls below the threshold; False Negatives (FNs) and True Negatives (TNs) are defined as their complementary cases.

III. EXPERIMENTAL SETUP

To evaluate the effectiveness of the proposed fault detection methodology, a series of fault injection experiments was conducted on a representative sample of CNNs trained on well-known image classification datasets.

The effectiveness of the proposed fault detection mechanism is evaluated on three widely used image classification datasets—MNIST, FashionMNIST, and CIFAR-10—covering a representative sample of CNN architectures. The selected networks include LeNet-5, MobileNetV2, and ResNet-18, each representing a different level of architectural complexity and parameter count. All models are trained from scratch, and their accuracy is evaluated in fault-free conditions. For CIFAR-10, the LeNet-5 architecture was slightly modified to increase its size, as the original version proved too simple for the dataset.

A statistical fault injection approach is adopted as described in [15], enabling controlled and reproducible injection of faults into trained DNNs. Random single bit-flips are injected into

the network weights, simulating the effects of permanent faults in memory and functional units [16].

The impact of injected faults is assessed by focusing on critical faults (CFs), defined as those causing at least one prediction within the test window to differ from the fault-free model. Such faults are considered to have a functional impact on the model's behavior. Table I shows the number of injected faults and the corresponding CFs for each CNN–dataset pair.

The auxiliary MLP of the proposed method was trained according to the following configurations. *Hidden layers*: 128 and 64 units; *Dropout*: 50%; *Optimizer*: Adam; *Learning rate*: 0.01; *Epochs*: 500; *Early stopping patience*: 20 epochs.

IV. EXPERIMENTAL RESULTS

Table I summarizes the performance of the proposed detection mechanism. For each CNN–dataset pair, the table reports the number of injected faults and the detection results in terms of TP, TN, FP, and FN. It also includes the number of raw features extracted from the first convolutional layer (# Feats. (raw)), the number of features retained after selection (# Feats. (selected)), the agreement threshold used, and the relative increase in inference time due to the detection mechanism.

The results show that the proposed fault detection mechanism provides outstanding detection rates. In faulty scenarios, the proposed system detects up to 97.79% (i.e., TP) of faults affecting the CNN operation, and at the same time, the false detection rates (i.e., FP) are maintained below 7%. It is worth noting that for certain CNN–dataset pairings, the detection capabilities can be reduced to 27.88%. Indeed, the results seem to indicate that the detection capabilities are significantly influenced by the evaluated dataset.

In detail, for the MNIST dataset, the mechanism achieves its best performance, with two out of three CNNs exceeding 78% of TP. Specifically, LeNet-5 reaches 97.79% of TP and ResNet-18 achieves 78.60%, while MobileNetV2 lags behind with 58.43%. The FP are 0.33%, 6.00%, and 7.33% for LeNet-5, MobileNetV2, and ResNet-18, respectively.

When evaluating the system on the FashionMNIST dataset, the results are more modest. In fact, LeNet-5 achieves 38.19% of TP, while MobileNetV2 and ResNet-18 reach 33.08% and 45.40% respectively. These lower TP values, with FN percentages exceeding 54% in all cases, can be attributed to the inherent visual similarity between certain classes (e.g., T-shirts and shirts), which could limit the discriminative potential of the extracted features and affect the fault detection process. Despite the modest fault detection capabilities, it is important to underline that the FP rates are the lowest across the evaluations, with values as low as 0.33%, 4.00%, and 6.00% for LeNet-5, MobileNetV2, and ResNet-18, respectively.

Finally, when considering the CIFAR-10 dataset, the results demonstrate strong fault detection capabilities for two out of the three evaluated CNNs. In particular, MobileNetV2 and ResNet-18 reach high TP rates of 79.73% and 81.82%, respectively, confirming the robustness and effectiveness of the proposed method even in the context of more complex data. These values are comparable to those observed on simpler

TABLE I
RESULTS OF THE FAULT DETECTION SYSTEM.

Dataset	CNN	Injected Faults	CF	TP (%)	FN (%)	TN (%)	FP (%)	Inc. Inf. Time (%)	Agreement Thresh. (%)	# Feats. (raw)	# Feats. (selected)
MNIST	LeNet-5	1,241	271	97.79	02.21	99.67	0.33	131.46%	90.00	140	33
MNIST	MobileNetV2	2,287	1,174	58.43	41.57	94.00	6.00	1.71%	94.00	660	125
MNIST	ResNet-18	3,418	458	78.60	21.40	92.67	7.33	4.21%	95.67	1,300	237
FMNIST	LeNet-5	1,241	927	38.19	61.81	99.67	0.33	74.45%	77.00	140	35
FMNIST	MobileNetV2	2,287	1,581	33.08	66.92	96.00	4.00	2.95%	84.33	660	102
FMNIST	ResNet-18	3,418	2,185	45.40	54.60	94.00	6.00	5.96%	86.00	1,300	189
CIFAR-10	LeNet-5	4,093	3,515	27.88	72.12	99.67	0.33	172.49%	59.33	660	225
CIFAR-10	MobileNetV2	4,093	513	79.73	20.27	97.33	2.67	9.84%	49.67	660	180
CIFAR-10	ResNet-18	5,100	671	81.82	18.18	97.33	2.67	22.37%	57.67	1,300	309

datasets, supporting the scalability of the detection mechanism when paired with sufficiently expressive models. The FP percentages for these architectures are also low, at 2.67% for both. In contrast, LeNet-5 performs poorly on this dataset, with a TP of only 27.88%. This result is likely due to the limited capacity of the network, which proves inadequate for managing the increased complexity of CIFAR-10, as also indicated by its lower classification accuracy of 75.61%. However, the FP for LeNet-5 remains low at 0.33%.

The increase in inference time introduced by the detection mechanism (column 9) is generally limited. For MobileNetV2 and ResNet-18 across all datasets, the overhead remains below 22.37%, with most values well under 10%, indicating a modest computational impact. Higher increases are observed for LeNet-5 (up to 172.5%), likely due to its small size, where the relative cost of feature extraction becomes significant. The memory required for the auxiliary MLP weights is negligible, never exceeding 190 KiB.

On the other hand, the results regarding the auxiliary MLP used by the fault detection system show that the number of extracted features is solely determined by the CNN architecture, as it depends on the size of the first convolutional layer's output feature maps (OFMs). However, the effectiveness of the feature extraction process also varies depending on the complexity of the dataset. As reported in the table, the fraction of selected statistical features ranges from 15% to 35%, with the number of selected features varying significantly across datasets. Notably, CIFAR-10 consistently requires a larger number of statistical features.

For example, 23.57%, 18.94%, and 18.23% of the statistical features are selected from the original set computed on the OFMs extracted from MNIST using LeNet-5, MobileNetV2, and ResNet-18, respectively. In the case of FashionMNIST, the selected feature fractions are 25.00% for LeNet-5, 15.45% for MobileNetV2, and 14.54% for ResNet-18. For CIFAR-10, the proportions of selected features are 34.09% for LeNet-5, 27.27% for MobileNetV2, and 23.77% for ResNet-18.

V. CONCLUSION AND FUTURE WORKS

This study proposes a detection system for identifying hardware faults in CNNs in real-time, without any modification to the underlying CNN. By processing multiple inputs (i.e., consecutive frames), the system determines the occurrence

of a fault by comparing the predictions performed with the early exit to those from the original CNN prediction. Our results indicate that detection performance depends on dataset complexity and network expressiveness, with deeper architectures generally achieving higher TP rates. In terms of detection effectiveness, the system achieves TP between 78.60% and 97.79% while keeping FP consistently below 7.33%. Future work will explore multi-layer monitoring and evaluate a broader selection of CNN–dataset pairs.

REFERENCES

- [1] H. Mun, *et al.*, "Recycling of adversarial attacks on the dnn of autonomous cars," in *Int. Conf. on Info. Networking (ICOIN)*, 2021, pp. 814–817.
- [2] P. H. Hochschild *et al.*, "Cores that don't count," in *Proc. of the Workshop on Hot Topics in Operating Systems*, 2021.
- [3] C. Schorn *et al.*, "Efficient on-line error detection and mitigation for deep neural network accelerators," in *Computer Safety, Reliability, and Security*, 2018, pp. 205–219.
- [4] P. Su *et al.*, "Using fault injection for the training of functions to detect soft errors of dnns in automotive vehicles," in *New Advances in Dependability of Networks and Systems*, 2022, pp. 308–318.
- [5] W.-K. Liu *et al.*, "Effective runtime fault detection for dnn accelerators," in *2024 IEEE Asian Test Symposium (ATS)*, 2024, pp. 1–6.
- [6] E. Ozen *et al.*, "Low-cost error detection in deep neural network accelerators with linear algorithmic checksums," *J. Electron. Test.*, vol. 36, no. 6, pp. 703–718, 2020.
- [7] H. F. Eniser *et al.*, "Deepfault: Fault localization for deep neural networks," in *Int. Conf. on Fundamental Approaches to Software Engineering*, 2019.
- [8] V. Turco *et al.*, "Early detection of permanent faults in dnns through the application of tensor-related metrics," in *2024 IEEE Int. Symp. on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, 2024.
- [9] A. Ye *et al.*, "Definder: Error-sensitive testing of deep neural networks via vulnerability interpretation," *J. Netw. Comput. Appl.*, vol. 241, p. 104212, 2025.
- [10] A. Ruospo *et al.*, "Image test libraries for the on-line self-test of functional units in gpus running cnns," in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–6.
- [11] G. Gavarini *et al.*, "Open-set recognition: an inexpensive strategy to increase dnn reliability," in *2022 IEEE Int. Symp. on On-Line Testing and Robust System Design (IOLTS)*, 2022, pp. 1–7.
- [12] Y. Matsubara *et al.*, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, 2022.
- [13] D. P. Kroese *et al.*, *Data Science and Machine Learning: Mathematical and Statistical Methods*. CRC Press, 2019.
- [14] P.-N. Tan *et al.*, *Introduction to Data Mining*. Addison-Wesley, 2005.
- [15] A. Ruospo *et al.*, "Assessing convolutional neural networks reliability through statistical fault injections," in *2023 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2023, pp. 1–6.
- [16] A. Bosio *et al.*, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*, 2019, pp. 1–6.