Investigating on Gradient Regularization for Testing Neural Networks

Nicolo Bellarmino, Alberto Bosio, Riccardo Cantoro, Annachiara Ruospo, Ernesto Sanchez, Giovanni Squillero

Investigating on Gradient Regularization for Testing Neural Networks

Nicolo' Bellarmino[1][0000−0001−5887−2598], Alberto Bosio[2][0000−0001−6116−7339], Riccardo Cantoro[1][0000−0002−1745−5293], Annachiara Ruospo[1][0000−0003−2040−9762], Ernesto Sanchez[1][0000−0002−7042−295X], and Giovanni Squillero[1][0000−0001−5784−6435]

[1] CAD Research Group
Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, 10129, IT
{nicolo.bellarmino,riccardo.cantoro,annachiara.ruospo,ernesto.sanchez}@polito.it
[2] Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France
alberto.bosio@ec-lyon.fr

Abstract. Convolutional Neural Networks (CNNs) have become ubiquitous in diverse applications, including safety-critical domains such as autonomous driving, where ensuring reliability is crucial. CNNs reliability can be jeopardized by the occurrence of hardware faults during the inference, leading to severe consequences. In recent years, gradient regularization has garnered attention as a technique able to improve generalization and robustness to Gaussian noise injected into the parameters of neural networks, but no study has been done considering its fault-tolerance effect. This paper analyzes the influence of gradient regularization on CNNs reliability for classification tasks in the presence of random hardware faults, exploring impacts on the network's performance and robustness. Our experiments involved simulating permanent stuck-at faults through statistical fault injection and assessing the reliability of CNNs trained with and without gradient regularization. Experimental results point out that regularization reduces the masking ability of neural networks, paving the way for efficient in-field fault detection techniques that aim at unveiling permanent faults. Specifically, it systematically reduces the percentage of masked faults up to 15% while preserving high prediction accuracy.

Keywords: Regularization · Neural Networks · Reliability · Hardware Faults · Gradient Regularization · Fault Tolerance · Machine Learning.

1 Introduction

Deep learning has revolutionized numerous fields, including image recognition, natural language processing, and autonomous systems. Deep neural networks (DNNs) and especially Convolutional Neural Networks (CNNs) are nowadays the most adopted standard Machine Learning (ML) models in the aforementioned fields (and beyond). In safety-critical applications such as autonomous

driving, the reliability of DNNs is paramount. Beyond mere performance metrics like accuracy, the robustness of these networks against unforeseen circumstances, particularly random hardware faults, is crucial. Hardware failures during inference can harm the system and have catastrophic consequences, like endangering human lives. Also, their architectures and high-dimensional parameter spaces pose challenges in ensuring their robustness and testability. Recent research has unveiled instances where DNN accuracy experiences notable declines in the presence of hardware faults [28, 20]. These findings underscore the importance of simultaneously evaluating DNN reliability.
In contemporary literature, gradient regularization (GR) has garnered significant interest owing to its potential for improving the accuracy of discriminative models and enhancing robustness against noise injected into the weights [3, 30]. In this study, our objective is to investigate the role of GR during the training phase of CNNs in enhancing their *reliability* and *testability* in classification tasks. Reliability in neural networks refers to the ability of the network to maintain its performance and accuracy under various conditions, such as the presence of noise. Testability, instead, refers to the ability to unmask possible hardware faults and let them propagate to the output (*observability*). From the literature [2, 4], it has been proven that CNNs inherently own a certain masking ability due to their redundancy. This is desirable in operational conditions, but it is disadvantageous when the CNN-based system undergoes a testing phase because detecting faults requires a long test time. In safety-critical systems, especially in automotive, a test procedure is periodically scheduled. It is therefore mandatory to detect the highest number of faults in the shortest time.
We review the current landscape of GR methods and their implications on model generalization and robustness. Experimental results showed that GR can increase the *testability* of the CNNs, facilitating the propagation of hardware faults to the output. The derived set of trained parameters, denoted as θ_{test}, can be used for the subsequent testing phase of the underlying hardware platform.
Our experiments involve two standard datasets for CNNs evaluation (MNIST and CIFAR-10), and 5 different architectures (a simple Le-Net-like CNN, two ResNet models, and two VGG models).

The rest of the paper is organized as follows: section 2 provides a review of related work on DNN resiliency; section 3 presents background information that describes theory and concepts useful for understanding successive experiments; section 4 describes the motivations and the proposed approach; section 5 presents the experimental setup, the dataset used, the ablation studies, and the overall results of the evaluations; finally, section 6 concludes the paper with the implications and limitations of the proposed approach, a summary of the main results, and future directions.

2 Related Work

Regularization plays a crucial role in preventing overfitting and improving the generalization performance of DNNs. However, its definition can vary, and dif-

ferent regularization methods are frequently investigated: [13] presents a systematic, unifying taxonomy to categorize existing methods, that may affect data, network architectures, error terms, and optimization procedures. Among the various regularization methods, gradient-based techniques have garnered significant attention [30, 3, 11, 32], being able to enhance model accuracy in supervised tasks [30, 32]. Also, it promotes the optimizer to find a minimum that resides within relatively flat regions of the optimization landscape [32], often leading to better generalization performance compared to sharp minima.

GR has been proven to be effective in ensuring robustness to both random and adversarial perturbations of the inputs of a DNN [17], and increasing robustness of DNN parameters in the presence of Gaussian additive or multiplicative noise [3, 6]. In [9], the authors utilized the gradient magnitude evolution throughout the model training process as a monitor to detect potential hardware faults during training. In [18], authors analyzed the effect of Max-Magnitude regularization in enhancing the resilience to single-bit-flit in quantized DNNs, finding that regularization methods did not improve the bit error resilience.

As for the DNN intrinsic masking effect, in [29] a fault detection technique based on evolutionary-based test images has been proposed to reduce the masking ability of neural networks.

In this work, the target is still reducing the masking ability of DNNs, but performed during the training phase by investigating the effect of gradient regularization in increasing the observability of DNNs for testing purposes.

3 Background

This section intends to provide background knowledge on the reliability assessment of DNNs (Section 3.1), with a focus on fault models (Section 3.2) and fault injection-based approaches (Section 3.3). Then, GR in DNNs is described in Section 3.4.

3.1 DNN Resiliency Analysis

The development of standards to address reliability concerns has become increasingly prevalent across various industries. One notable example is the *ISO 26262* standard [1], which is widely adopted in the automotive sector to ensure functional safety in the design and production of vehicles. Similarly, new standards focusing on the functional safety of AI systems, such as *ISO/IEC CD TR 5469* [10], are being introduced to guide this rapidly evolving domain. From a functional point of view, DNNs can be seen as software programs executed on a given hardware, and thus, can be analysed by functional approaches. Functional testing methods rely on the comparison of outcomes between a faulty system and the same system operating under fault-free conditions. A trained DNN, ready to be deployed in the field, can be tested with functional methods by inspecting its outputs in the possible presence of faults. DNNs are known to possess an intrinsic fault tolerance due to their extensive and redundant interconnected

layers. The high number of parameters enables them to mitigate a certain degree of possible hardware-faults [2, 4]. However, in the literature [23] has been shown that in safety-critical applications a single fault may lead to catastrophic consequences, e.g., an autonomous driving car that misclassifies a pedestrian crossing the road.

3.2 Fault models

Faults affecting electronic devices can be categorized into two main types: permanent and transient. Permanent faults typically result from irreversible physical damage to the electronic components. They represent enduring defects that cannot be rectified through normal operation. Examples include manufacturing defects, material degradation, or physical damage due to environmental factors. Transient faults, instead, are temporary faults that arise from external disturbances or abnormal conditions/events. Among the various sources are interference phenomena, such as high-energy particle strikes, and electrical or electromagnetic noise. Transient faults are also known as soft errors and can affect registers or memory regions within electronic devices. The random bit-flip model is often used to represent transient faults, where bits in memory or registers flip spontaneously due to external factors.

In the DNN reliability field, the stuck-at fault model [22] is a widely used abstraction of permanent faults, as many transistor and interconnection defects can be accurately represented as permanent defects at the logic level. Specifically, a widely recognized practice consists of corrupting individual bits in the weights of neural networks. This effect mimics the occurrence of random physical hardware faults in memories (that could be permanent or transient) since during inference weights are treated as constant data and stored in memories.

To conduct reliability studies, the effect of faults affecting the underlying hardware is typically categorized into three primary categories [22]: *Masked*, *Non-Critical*, and *Critical* (also known as *Silent Data Corruption SDC-1*). The first category includes all those faults that keep the prediction correct, and that do not change the output vector of the scores; in other words, the output of the faulty DNN remains identical to the fault-free one. Essentially, the fault is present but does not lead to any change in the output. It is not *observable*, or it is not propagated in the output of the network. For the second category, the output of the faulty DNN differs from the fault-free output, but the variation is still deemed acceptable by the end user. As an example, for a classification system, the predicted probability associated with the different classes considered changes between faulty and faulty-free DNNs, but the predicted class (i.e. the $argmax$ of the probabilities) remains the same. A critical fault, instead, results in output changes that are not acceptable to the end user [16]. The deviations from the fault-free output are significant and impact the usability or reliability of the system. For a classification system, the predicted class between faulty and faulty-free DNNs changes.

3.3 Fault Injection

Among the different dependability testing techniques, fault injection (FI) is a widely used and powerful method that involves introducing errors (or *faults*) into a system to observe their impact on the system behavior. Depending on the abstraction level, FI methods can be categorized into three main types [26]: *Simulation-based*, *Platform-based*, and *Radiation-based* FIs. Simulation-based FIs operate on a model of the device described using a simulation language. Faults can be injected into the architectural representation of the hardware model (e.g., the HDL model) either at run-time or at compile-time, or they can be introduced into the software level of the application. Software-level simulation-based FIs injects faults directly into program variables or instructions without considering the hardware layer. It has low implementation costs, high controllability, and observability. However, they may produce non-realistic results due to the lack of knowledge about the underlying hardware. In platform-based FIs, the measurements and analyses are performed directly on a physical device that emulates the final implementation of a design using FPGAs or on physical platforms that run DNNs, for example, CPUs and GPUs. They have a medium cost, requiring the use of validation or emulation devices such as GPUs, CPUs, and FPGAs. Once purchased, the advantage is that they can be reused after FI campaigns. Finally, as for radiation-based FIs, the actual implementation of the system is exposed to the same external conditions as the in-field application (for example, a flux of atmospheric-like neutrons, which can induce single-event effects on electronic devices). These FI experiments lead to very accurate results, but are extremely expensive in terms of equipment and facility access, and can only be performed late in the design process when the physical device is available.

Evaluating the reliability of a DNN may involve running a complete *FI campaign*, and thus considering every possible bit-flip fault of every bit within the data representation of weights, typically in 32-bit floating-point format (FP32). To do this, it is common to use the entire validation/test set. The main problem of this approach is the time requirement: we can take as an example a test set composed of 10,000 images and a relatively small set of faults (100,000) to be injected. In such a scenario, the experiment would necessitate conducting 10^8 fault injections. Assuming a reasonable execution time of 1 ms per fault injection in a relatively small CNN (e.g. ResNet20), the time required would be approximately 12 days. This exhaustive process becomes increasingly impractical with larger and more intricate models, in which the number of weights usually reaches millions. To address this challenge, research works have proposed to exploit statistical sampling to reduce the fault space, i.e., the injection of a reduced set of faults [25]. Statistical FI campaigns can be used to obtain statistically significant results when assessing the reliability of a DNN, without the need for exhaustive fault simulations. This procedure is based on extracting a subset of all the possible faults (representative and statistically significant) that may happen in a targeted DNN (and a targeted population of faults). Experimental results show that by injecting a reduced quantity of faults (e.g., 0.55% on

MobileNetV2 trained and tested on CIFAR-10 [25]) it is possible to achieve an estimate with a maximum error margin of 1% and a confidence of 99%.

3.4 Gradient Regularization

Training a neural network involves minimizing a loss function L, computed on a dataset $D(X, y)$ with respect to the parameters of the model θ. The loss function L measures the difference between the predicted output $\hat{y}$ of the neural network and the ground truth label y for a given input X. The goal is to find the optimal parameters θ that minimize the loss function, which corresponds to the best fit of the neural network to the dataset. This is a non-convex optimization problem [3], carried on using optimization algorithms, such as stochastic gradient descent (SGD), Adam, AdaDelta [24, 5, 31], which iteratively update the parameters θ to minimize the loss function. The choice of loss function L depends on the specific problem and the type of neural network being used. For example, for a classification problem, a common choice is the cross-entropy loss [19].

Regularization techniques play a vital role in improving model generalization [13, 32], encompassing various approaches such as penalty function methods, data augmentation, dropout regularization, normalization techniques, and more. Penalty function methods involve adding extra terms to the loss function and optimizing them alongside it, to impose constraints on specific properties of models, in the form of:

$$L'(\theta; X, y) = L(\theta; X, y) + \alpha P(\theta) \tag{1}$$

In which L' is the loss function to minimize, computed on inputs X and label y with respect to the model parameters θ. L is the loss function related to the supervised task we aim to solve, such as the cross-entropy loss in classification problems.

GR has indeed garnered significant attention from researchers in recent years [32, 30, 3, 11]. Including a penalty term computed on the gradient of the loss functions with respect to the model weights could generally improve the generalization capabilities and the stability of the learning procedure of a neural network [32]. The loss function (already studied [3]) is thus:

$$L'(\theta; X, y) = L(\theta; X, y) + \alpha ||\nabla L(\theta; X, y)||^2 \tag{2}$$

Penalizing large gradient norms could lead to flat minima in the loss surface, regions in which parameters are more robust regarding perturbations after training (in the form of multiplicative or additive Gaussian noise [3, 6])

4 Methodology

Numerous studies have explored the robustness of DNNs in the presence of parameter noise, often finding that DNNs exhibit considerable resilience, particularly in networks with higher test accuracy [21, 3]. GR has emerged as a technique

to bolster this robustness against perturbations post-training. However, existing investigations primarily focus on Gaussian additive or multiplicative noise, rather than simulated hardware faults, which mimic real-world deployment scenarios where DNNs operate as applications on specific hardware platforms, such as GPUs or dedicated chips. Our study aims at addressing this gap by conducting a rigorous analysis of DNNs trained using the loss function described in Equation 2, specifically *evaluating the impact of gradient regularization on faults affecting single bits of the network parameters.* In a dependable neural network, the impact on the output of a hardware fault should be minimized as much as possible. Thus, *Critical* faults should be minimized, while the faults classified as *masked* should be maximized. This property is highly desirable in safety-critical environments, as it ensures that decisions or analyses based on the network's output remain unaffected even when faults occur. However, *having a high number of masked faults means reducing the observability of the network*, thus making it more difficult to unveil the presence of a fault in the underlying output with functional methods. Hence, depending on the system, task, and ultimate objective, engineers can strive to either maximize or minimize masked faults. In real-world scenarios, it would be feasible to maintain a dual set of parameters, denoted as θ and θ_t, for the DNN: one (θ) optimized to achieve the highest degree of fault masking for deployment in field inference, and the other (θ_t) designated for testing the underlying hardware in an online test environment.

By simulating hardware faults, we aim to provide insights into the reliability and testability of DNNs in realistic deployment environments, where the integrity of individual parameter bits can significantly influence performance and safety. This approach enables us to assess the effectiveness of gradient regularization in mitigating (or not) the effects of such faults.

5 Experiments

5.1 Datasets

To validate our experiments we used CIFAR-10 [12] and MNIST [15] datasets. These usually serve as cornerstone benchmarks in computer vision and machine learning, facilitating the evaluation and comparison of various algorithms and architectures. CIFAR-10 comprises 60,000 32x32 color images across ten classes, while MNIST consists of 60,000 28x28 grayscale images representing handwritten digits from zero to nine. For both dataset, the test set is composed by 10,000 images. Images were pre-processed before being fed to the DNN. For CIFAR-10, we normalize each channel using predefined mean and standard deviation ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)). For MNIST, image transformations included resizing to a 32x32 resolution (to standardize DNN architectures between datasets) and normalizing the image data using predefined mean and standard deviation values (0.1307, 0.3081). We used data augmentation during the training to reduce the overfitting of the networks, applying random affine transformations such as rotation, translation, and scaling.

5.2 Models

Our analysis considered five models across three distinct architectures. Residual Neural Networks (ResNets) [8] represents a pivotal advancement in deep learning architecture. By introducing innovative identity skip connections, commonly known as "residual connections", they target the issue of degradation in training accuracy with increased network depth. Skip Connections facilitate the training of deep networks comprising tens or hundreds of layers, preventing the vanishing gradient problem [8] inherent in deep networks. We used two ResNet architectures with 9 and 18 layers. LeNet [14] is a pioneering CNN architecture that has been influential in the field of deep learning, particularly in the context of document recognition and image processing tasks We used a Le-Net style CNN that consists of two convolutional layers with ReLU activation functions, followed by max pooling, dropout, flattening, and two fully connected layers, culminating in a final linear layer for classification. Finally, the VGG [27] architecture. It is characterized by uniformity with consistent use of 3x3 filters in the convolutional layers and a maximum pooling layer for downsampling after each block of convolutional layers. We used VGG11 and 16, respectively with 11 and 16 layers.

5.3 Parameters Setting

In our experimental setup, we employed the Adadelta optimizer [31]. For the CIFAR-10 dataset, we conducted training over 100 epochs, while for MNIST, we trained for 20 epochs. These epochs revealed being enough to reach near state-of-the-art testing accuracy with all the networks considered. A batch size of 64 was utilized for both datasets. We initialized the learning rate at 1 and incorporated a learning rate step-decay strategy, with $\gamma = 0.9$ the step=5 for CIFAR-10, while we used $\gamma = 0.7$ with a step=3 for MNIST. Cross Entropy Loss [19] was employed, in conjunction with gradient regularization as necessary. Our experiments were conducted using PyTorch, leveraging its efficient *autograd* functionality for gradient computation. Faults Injection campaign was conducted using statistical fault injection [25], with a dedicated Python framework to perform FI campaign on NVIDIA-CUDA devices [7].

Statistical FI makes it feasible to run an FI campaign on DNNs in a reasonable amount of time by computing the sample size, denoted as n required to reach the maximum error of an estimate.

The formula for calculating the sample size (n) given the maximum error of the estimate (e) is derived as follows [25]:

$$e = t \cdot \frac{\sigma}{\sqrt{n}} \tag{3}$$

This equation can be rearranged to solve for n:

$$n = \frac{N}{1 + \frac{e^2 \cdot (N-1)}{t^2 \cdot p \cdot (1-p)}} \tag{4}$$

In which e represents the desired error margin; t, a constant, is the critical value associated with the desired confidence level; σ denotes the population standard deviation; N is the population size; p is the probability of a bit-flip to happen. We used $p = 0.5$, $e = 0.01$, $t = 2.58$, while N is the total number of bit-flips that may happen, thus the total number of parameters in the DNN multiplied by 32 (FP32 representation) multiplied by 2 (0-to-1 and 1-to-0 transitions). For each model under consideration, Eq. 4 leads to about 16,660 faults to inject to obtain an estimate with a maximum error margin of 1%. The number of faults to inject into each layer of the DNN, l, is determined by calculating the proportion of the total faults relative to the parameters of that specific layer. This is computed by rounding the product of the number of parameters in the layer l and the desired total number of faults to inject, divided by the total sum of parameters. This distributes the faults to the layers based on their respective parameter counts. Then, for each layer, random parameters and random bits from these are selected until the desired number of faults is reached. The selected bit is then flipped. The FI campaign was based on applying every fault in the fault list to the DNN and running the inference on the chosen test set. For each output vector, we compare it with the fault-free execution, marking each fault as masked, critical or not critical depending on the comparison outcome. The number of outputs for each FI campaign is thus the dimensionality of the test set multiplied by the number of faults applied to each network (see Tab. 1). The FI campaign procedure is explained in Alg. 1

As model performance metrics, we used the baseline accuracy and how it is affected by faults, as well as the distribution of different types of faults (masked, not critical, critical). The *Clean Accuracy* is the baseline accuracy of the network on the test set. It is informative on how likely the network will perform the correct classification without faults. The *Faulty Accuracy* is the accuracy of the network in the presence of a fault. It is computed as the number of times the network predicted the correct class divided by the number of faulty outputs analyzed. It is informative on how likely the network will perform the correct classification when a fault occurs. Percentages of distribution of faults reflect the number of outcomes in which a fault was marked masked, critical, or not critical. These are informative on how likely a fault is propagated to the output or not and its effect on the classification outcome.

Values of α parameter were selected by trial and error: depending on the model, high values lead to an over-regularization, and thus to the divergence of the training procedure. Therefore, we used values that permitted sticking, with a certain degree, to baseline accuracy achieved without regularization.

5.4 Results

The main effect of GR is a shift in the distribution of fault categories (Tables 2 and 3). The number of masked-faults decreases. We have this effect for both the dataset and for all the networks taken into account. The number of *critical* faults is quite similar among default and regularized loss, with a slight increase for some architecture. Notably, a significant portion of faults initially labeled

Table 1. FI campaign information for the different networks

Network	Dataset	#Parameters	#Possible Faults	Applied Faults	Outcome
ResNet18	CIFAR-10	11,173,962	715,133,568	16,641	16,641*10^4
ResNet9	CIFAR-10	2,440,266	156,177,024	16,639	16,639*10^4
LeNet	CIFAR-10	1,626,442	104,092,288	16,638	16,638*10^4
VGG11	CIFAR-10	9,756,426	624,411,264	16,641	16,641*10^4
VGG16	CIFAR-10	15,253,578	976,228,992	16,641	16,641*10^4
ResNet18	MNIST	11,172,810	715,059,840	16,641	16,641*10^4
ResNet9	MNIST	2,439,114	156,103,296	16,639	16,639*10^4
LeNet	MNIST	1,626,442	104,092,288	16,638	16,638*10^4
VGG11	MNIST	9,755,274	624,337,536	16,641	16,641*10^4
VGG16	MNIST	15,252,426	976,155,264	16,641	16,641*10^4

Table 2. Accuracy metrics and fault-type distributions (CIFAR-10)

Model	Loss	Clean Acc	Faulty Acc	Masked	Not Critical	Critical
ResNet18	Default	93.52%	91.24%	30.80%	66.72%	2.46%
ResNet18	Gradient ($\alpha = 0.01$)	93.48%	91.14%	28.46%	69.02%	2.52%
ResNet9	Default	92.17%	89.64%	23.03%	74.19%	2.77%
ResNet9	Gradient ($\alpha = 0.05$)	91.55%	88.96%	21.81%	75.32%	2.86%
LeNet	Default	79.36%	79.07%	91.28%	8.36%	0.34%
LeNet	Gradient ($\alpha = 0.05$)	78.04%	77.47%	82.22%	17.01%	0.76%
VGG11	Default	90.10%	88.15%	50.11%	47.70%	2.18%
VGG11	Gradient ($\alpha = 0.01$)	90.37%	88.32%	43.49%	54.22%	2.29%
VGG16	Default	91.61%	89.53%	41.32%	56.38%	2.29%
VGG16	Gradient ($\alpha = 0.005$)	91.62%	89.47%	35.91%	61.72%	2.36%

as *masked* now are recognized as *not critical*. Resulting networks are not more resilient or robust: in the presence of faults, the percentage of outcomes that differ between clean and faulty inference is increased. However, this does not lead to changing the predicted class (the not-critical faults remains practically the same). Surprisingly, GR reduced the resilience and robustness of DNN to bit flips perturbation. However, it made them more *observable*: if the percentage of not-critical increases, it is easier for the effect of a fault to be propagated to the output and thus be visible. This property is desirable for testability purposes since it increases the number of faults in the hardware that can be identified by looking at the outcome of the network. This reduction can indicate improved fault detection or mitigation strategies, leading to a higher level of transparency and understanding of DNN systems vulnerabilities. This property can be used in addition to existing fault detection methods, to obtain an additional set of weights θ_t to be used for online functional testing of underlying hardware. This effect is limited in some cases (ResNets, CIFAR-10) but is non-negligible in

Algorithm 1 Fault Injection Campaign

Require: n: number of faults to inject
Require: DNN: NN to analyze
  $faultList \leftarrow [\,]$
  $N \leftarrow model.getParamsNum$ ▷ Generate the Fault List
  for $layer \in model.layersList$ **do** ▷ For each layer:
    $i \leftarrow 0$
    $e = (layer.getParamsNum/N) * n$ ▷ Compute faults to inject into each layer
    while $i \leq e$ **do**
      $\theta_l = randomSelectParameter(layer)$
      $b = randInt(0, 31)$ ▷ Selection of a bit to flip
      $faultList.append(l, \theta_l, b)$ ▷ Append to the Fault List
    end while
  end for
  for $f \in faulList$ **do** ▷ Apply every fault in the faults list
    $D\hat{N}N = copy(DNN)$
    $D\hat{N}N.applyFault(f)$
    for $testImage \in testSet$ **do**
      $faultyOutput = D\hat{N}N(testImage)$
      $goldenOutput = DNN(testImage)$
      $response = categorizeFault(goldenOutput, faultyOutput)$
      $save(f, testImage, response)$
    end for
  end for

others (ResNet9 on MNIST and LeNet on CIFAR-10, with a decrease in masked faults between 9% and 12%)

5.5 Ablation Studies

We furthermore studied the effect of GR with ablation studies on subsets of models and datasets, to validate our experiments. These involved altering three key components: the optimizer employed to train the network, the regularization strength parameters α and the epochs of training.

Changing the optimizer We performed these experiments on CIFAR-10 dataset for the two ResNet architectures. We replaced AdaDelta optimizer with Adam. We chose learning rate $lr = 0.01$. Higher values lead to divergence in the training procedure, while lower values to a increase in convergence time. We run the network optimization for 150 epochs. We found that the choice of optimizer impacts the DNN reliability: the number of masked faults with ADAM optimizer (see Tab 4) dramatically increases. The obtained networks are thus more resilient to be used in the field. In this case, the effect of the gradient regularization is to increase the overall accuracy of the model, in accordance with the premises. Also in this case, the number of masked faults decreases if gradient regularization is

Table 3. Accuracy metrics and fault-type distribution (MNIST)

Model	Loss	Clean Acc	Faulty Acc	Masked	Not Critical	Critical
ResNet18	Default	99.63%	97.08%	25.10%	72.34%	2.55%
ResNet18	Gradient ($\alpha = 0.1$)	99.49%	96.96%	24.70%	72.76%	2.53%
ResNet9	Default	99.65%	96.87%	17.82%	79.39%	2.78%
ResNet9	Gradient ($\alpha = 0.05$)	99.61%	96.85%	1.62%	95.59%	2.78%
LeNet	Default	99.25%	99.09%	95.79%	4.05%	0.15%
LeNet	Gradient ($\alpha = 0.06$)	99.19%	98.89%	92.76%	6.93%	0.30%
VGG11	Default	99.68%	97.36%	42.06%	55.60%	2.33%
VGG11	Gradient ($\alpha = 0.05$)	99.63%	97.31%	39.88%	57.78%	2.33%
VGG16	Default	99.58%	97.22%	38.15%	59.47%	2.36%
VGG16	Gradient ($\alpha = 0.08$)	99.51%	97.03%	32.45%	65.80%	2.49%

Table 4. Effect of using ADAM optimizer, CIFAR10

Model	Loss	Clean Acc	Faulty Acc	Masked	Not Critical	Critical
ResNet18	Default	90.50%	89.50%	90.68%	8.20%	1.12%
ResNet18	Gradient ($\alpha = 0.1$)	91.47%	90.36%	89.05%	9.71%	1.23%
ResNet9	Default	81.06%	79.18%	90.76%	6.85%	2.37%
ResNet9	Gradient ($\alpha = 0.05$)	83.87%	81.84%	88.84%	8.67%	2.48%
VGG11	Default	88.37%	86.72%	62.82%	35.30%	1.88%
VGG11	Gradient ($\alpha = 0.05$)	89.23%	87.53%	54.82%	43.24%	1.93%

used, even if the percentages of only 1-2% for the ResNets and up to 8% for VGG.

Varying the α parameter We performed these experiments on MNIST dataset, considering the VGG11, VGG16 and ResNet18 architectures. We used AdaDelta optimizer. Increasing the α seemed to impact the percentage of masked faults (Tab. 5). By starting from 'low' values of α, the number of masked faults seems to be higher with respect to default cross-entropy loss. However, increasing α leads to a reduction in the percentage of masked faults. Regularization strength has thus an impact.

Varying the training epochs We performed these experiments on CIFAR-10 dataset, considering the LeNet architecture. In previous experiments, this network showed the highest masking capabilities. Also in this experiment, gradient regularization decreases the masked faults by a non-negligible percentage (7-9%, see Tab 6).

Table 5. Effect of varying α, MNIST

Model	Loss	Clean Acc	Faulty Acc	Masked	Not Critical	Critical
ResNet18	Gradient ($\alpha = 0.005$)	99.63%	97.12%	26.05%	71.42%	2.52%
ResNet18	Gradient ($\alpha = 0.01$)	99.61%	97.09%	25.11%	72.35%	2.52%
ResNet18	Gradient ($\alpha = 0.1$)	99.49%	96.96%	24.70%	72.76%	2.53%
ResNet9	Gradient ($\alpha = 0.025$)	99.64%	96.87%	19.61%	77.60%	2.78%
ResNet9	Gradient ($\alpha = 0.03$)	99.66%	96.89%	19.73%	77.48%	2.78
ResNet9	Gradient ($\alpha = 0.05$)	99.61%	96.85%	1.62%	95.59%	2.78%
VGG11	Gradient ($\alpha = 0.01$)	99.71%	97.42%	43.41%	54.29%	2.29%
VGG11	Gradient ($\alpha = 0.03$)	99.67%	97.40%	44.90%	52.80%	2.28%
VGG11	Gradient ($\alpha = 0.05$)	99.63%	97.31%	39.88%	57.78%	2.33%
VGG16	Gradient ($\alpha = 0.005$)	99.66%	97.37%	39.29%	58.40%	2.30%
VGG16	Gradient ($\alpha = 0.01$)	99.63%	97.27%	37.82%	59.80%	2.37%
VGG16	Gradient ($\alpha = 0.08$)	99.51%	97.03%	32.45%	65.80%	2.49%

Table 6. Effect of varying the epochs of training

Model	Loss	Clean Acc	Faulty Acc	Masked	Not Critical	Critical
LE NET (50 EPOCHS)	Default	78.04%	77.47%	82.22%	17.01%	0.76%
LE NET (50 EPOCHS)	Gradient ($\alpha = 0.06$)	75.15%	74.42%	75.84%	23.14%	1.01%
LE NET (30 EPOCHS)	Default	72.61%	74.35%	90.19%	9.42%	0.38%
LE NET (30 EPOCHS)	Gradient ($\alpha = 0.06$)	72.97%	72.43%	81.35%	17.87%	0.78%

6 Conclusion

We compared DNNs trained with cross-entropy loss versus cross-entropy loss with gradient regularization to assess if the latter impacts reliability in the presence of simulated hardware faults. While gradient regularization did not increase DNN reliability, as evidenced by the unchanged number of critical faults and not an increase in masked faults, it did demonstrate a notable capability to reduce the occurrence of masked faults. The impressive reductions observed in certain cases, such as with ResNet9 on Cifar 10 and LeNet architectures, were particularly noteworthy. Although this characteristic may not directly enhance NN reliability, it offers substantial potential for improving testability, which is crucial for the functional testing of DNNs. Moreover, our ablation showed a consistent decrease in masked faults when gradient regularization was used. This technique could be used to obtain a set of parameters θ_t usable for testability of underlying hardware, thus providing valuable insights for optimizing NN performance and robustness in safety-critical applications. Future work aims to study how changing the training setup affects the DNNs reliability.

Acknowledgment

This work has been funded by the ANR France 2030 AdapTING project "ANR-23-PEIA-0009".

References

1. Road vehicles – functional safety (2011)
2. Ahmadilivani, M.H., Barbareschi, M., Barone, S., Bosio, A., Daneshtalab, M., Torca, S.D., Gavarini, G., Jenihhin, M., Raik, J., Ruospo, A., Sanchez, E., Taheri, M.: Special session: Approximation and fault resiliency of dnn accelerators. In: 2023 IEEE 41st VLSI Test Symposium (VTS). pp. 1–10 (2023). https://doi.org/10.1109/VTS56346.2023.10140043
3. Barrett, D.G.T., Dherin, B.: Implicit gradient regularization (2022)
4. Bosio, A., Bernardi, P., Ruospo, A., Sanchez, E.: A reliability analysis of a deep neural network. In: 2019 IEEE Latin American Test Symposium (LATS). pp. 1–6 (2019). https://doi.org/10.1109/LATW.2019.8704548
5. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization Methods for Large-Scale Machine Learning. Society for Industrial and Applied Mathematics (SIAM) Review **60**(2), 223–311 (2018). https://doi.org/10.1137/16M1080173, https://doi.org/10.1137/16M1080173
6. Dey, P., Nag, K., Pal, T., Pal, N.: Regularizing multilayer perceptron for robustness. IEEE Transactions on Systems, Man, and Cybernetics: Systems **PP** (02 2017). https://doi.org/10.1109/TSMC.2017.2664143
7. Gavarini, G., Ruospo, A., Sanchez, E.: Sci-fi: a smart, accurate and unintrusive fault-injector for deep neural networks. In: 2023 IEEE European Test Symposium (ETS). pp. 1–6 (2023). https://doi.org/10.1109/ETS56758.2023.10173957
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
9. He, Y., Hutton, M., Chan, S., De Gruijl, R., Govindaraju, R., Patil, N., Li, Y.: Understanding and mitigating hardware failures in deep learning training systems. In: Proceedings of the 50th Annual International Symposium on Computer Architecture. ISCA '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3579371.3589105, https://doi.org/10.1145/3579371.3589105
10. ISO and IEC: Artificial intelligence — functional safety and ai systems. Technical Report 5469, ISO/IEC JTC 1/SC 42, Geneva, CH (01 2024), https://standards.iteh.ai/catalog/standards/iso
11. Karakida, R., Takase, T., Hayase, T., Osawa, K.: Understanding gradient regularization in deep learning: Efficient finite-difference computation and implicit bias (2023)
12. Krizhevsky, A.: Cifar-10 dataset. https://www.cs.toronto.edu/ kriz/cifar.html (2009), accessed on April 11, 2024
13. Kukačka, J., Golkov, V., Cremers, D.: Regularization for deep learning: A taxonomy (2017)
14. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)

15. LeCun, Y., Cortes, C., Burges, C.J.: The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/ (1998), accessed on April 11, 2024
16. Li, G., Hari, S.K.S., Sullivan, M., Tsai, T., Pattabiraman, K., Emer, J., Keckler, S.W.: Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In: SC17: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–12 (2017)
17. Li, Q., Hu, Q., Lin, C., Wu, D., Shen, C.: Revisiting gradient regularization: Inject robust saliency-aware weight bias for adversarial defense. IEEE Transactions on Information Forensics and Security **18**, 5936–5949 (2023). https://doi.org/10.1109/TIFS.2023.3289000
18. Liu, Z., Yang, X.: An efficient structure to improve the reliability of deep neural networks on arms. Microelectronics Reliability (2022), https://api.semanticscholar.org/CorpusID:251485318
19. Mao, A., Mohri, M., Zhong, Y.: Cross-entropy loss functions: Theoretical analysis and applications (2023)
20. Mittal, S.: A survey on modeling and improving reliability of dnn algorithms and accelerators. J. Syst. Archit. **104**(C) (mar 2020). https://doi.org/10.1016/j.sysarc.2019.101689, https://doi.org/10.1016/j.sysarc.2019.101689
21. Morcos, A.S., Barrett, D.G.T., Rabinowitz, N.C., Botvinick, M.: On the importance of single directions for generalization (2018)
22. Pappalardo, S., Ruospo, A., O'Connor, I., Deveautour, B., Sanchez, E., Bosio, A.: A fault injection framework for ai hardware accelerators. In: 2023 IEEE 24th Latin American Test Symposium (LATS). pp. 1–6 (2023). https://doi.org/10.1109/LATS58125.2023.10154505
23. Qian, C., Zhang, M., Nie, Y., Lu, S., Cao, H.: A survey of bit-flip attacks on deep neural network and corresponding defense methods. Electronics **12**(4) (2023), https://www.mdpi.com/2079-9292/12/4/853
24. Ruder, S.: An Overview of Gradient Descent Optimization Algorithms. arXiv (2016)
25. Ruospo, A., Gavarini, G., de Sio, C., Guerrero, J., Sterpone, L., Reorda, M.S., Sanchez, E., Mariani, R., Aribido, J., Athavale, J.: Assessing convolutional neural networks reliability through statistical fault injections. In: 2023 Design, Automation and Test in Europe Conference I& Exhibition (DATE). pp. 1–6 (2023). https://doi.org/10.23919/DATE56975.2023.10136998
26. Ruospo, A., Sanchez, E., Luza, L.M., Dilillo, L., Traiola, M., Bosio, A.: A survey on deep learning resilience assessment methodologies. Computer **56**(2), 57–66 (2023)
27. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
28. Su, F., Liu, C., Stratigopoulos, H.G.: Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives. IEEE Design & Test **40**(2), 8 – 58 (Apr 2023). https://doi.org/10.1109/MDAT.2023.3241116, https://hal.science/hal-03961502
29. Turco, V., Ruospo, A., Gavarini, G., Sanchez, E., Reorda, M.S.: 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). pp. 1–6 (2023)
30. Varga, D., Csiszárik, A., Zombori, Z.: Gradient regularization improves accuracy of discriminative models (2018)
31. Zeiler, M.D.: Adadelta: An adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)

32. Zhao, Y., Zhang, H., Hu, X.: Penalizing gradient norm for efficiently improving generalization in deep learning. ArXiv **abs/2202.03599** (2022), https://api.semanticscholar.org/CorpusID:246652190