Strategies for Improving the Error Robustness of Convolutional Neural Networks

António Morais[1], Raul Barbosa[1], Nuno Lourenço[1], Frederico Cerveira[1],
Michele Lombardi[2], and Henrique Madeira[1]
[1]University of Coimbra, CISUC, DEI, Coimbra, Portugal
[2]University of Bologna, DISI, Bologna, Italy

***Abstract*—The error robustness of Convolutional Neural Networks (CNNs) is an important attribute requiring attention due to their growing application in safety-critical domains such as autonomous driving and medical devices. Hardware errors affecting the execution of such models may lead to system failures and, therefore, fault tolerance techniques are necessary to improve dependability. This paper proposes an approach to improve the robustness of CNNs and experimentally compares it with three other existing techniques. Fault injection is used to emulate hardware faults affecting CNNs targeting four distinct datasets. Results indicate that the ranger technique globally provides the best robustness closely followed by the stimulated training technique, although the former provides much lower temporal overhead than the latter. Architectural redundancy and dropout provide varying results. In all cases, caution through final evaluation of any CNN is required, because there are corner cases in which the robustness decreases, contrary to the intended outcome.**

***keywords*—Dependability, neural networks, hardware faults, safety.**

I. Introduction

The increasing usage of Machine Learning (ML) techniques in safety-critical applications raises concerns about the dependability of models and algorithms due to their non-deterministic and probabilistic outputs, limitations of the training data, and non-determinism in testing that makes it virtually impossible to cover all corner cases. Therefore, to harness the benefits of using such techniques in scenarios like autonomous driving and medical devices, one must devise adequate mechanisms to ensure that they are dependable.

When using CNNs, failures are often caused by inadequate network architectures, insufficient training data or overfitting. Moreover, hardware faults also represent an important threat because CNNs usually require massive hardware accelerators that are susceptible to soft errors [1], [2]. Radiation can upset internal states of circuits and cause data errors while the hardware itself remains undamaged [3] and therefore such events are called "soft" errors. Previous studies [4], [5] have observed significant corruptions in the outputs of neural networks, caused by soft errors. Such errors may affect the execution and lead to classification failures, and therefore the error robustness of CNNs must be assessed and improved to allow for their use in safety-critical scenarios.

This paper proposes a novel fault tolerance technique named Stimulated Training, evaluates the effectiveness of the existing Dropout technique with regards to error robustness, and compares the effectiveness of two other existing fault tolerance methods to improve CNNs robustness. Specifically, we consider four distinct techniques: Redundancy, Ranger, Dropout and Stimulated Training. We measure the error robustness of neural networks through fault injection at the ISA (Instruction Set Architecture) level, using the ucXception framework [6], [7]. Fault injection is performed during the testing phase of neural network models and metrics like Silent Data Corruptions (SDCs) and accuracy are collected to characterize the results of each technique.

Generally speaking, neural networks are regarded as having intrinsic robustness against errors. However, such intrinsic redundancy is likely insufficient for safety-critical applications and requires adequate validation. To this end, the paper makes the following contributions:

- Dropout, which is a well known regularization technique used to avoid overfitting [8], is examined for the hypothesis of also improving error robustness. To the best of our knowledge, this is the first study of this nature. The technique works by randomly cutting off neurons and, intuitively, force the network to learn to adapt. This, in principle, could have an effect on soft error sensitivity. However, in practice, we observe a limited effect.
- Based upon the results of the dropout strategy, we devise a new technique named Stimulated Training. The new method consists of emulating single bit flip errors during training (instead of simply cutting neurons off). If neurons were fail-silent, then the original dropout technique could potentially work well. However, as neurons are non-fail-silent, the Stimulated Training technique is proposed and experimentally observed to provide better results than dropout.
- The paper presents the results of an experimental evaluation including two other existing techniques for improving the robustness of CNNs, totalling 43 campaigns and $371\,520$ faults injected. Therefore, four techniques are globally compared: redundancy [9], ranger [10], dropout [8] and the novel stimulated training.

The remainder of the paper is organized as follows. Section II describes related work and background related to three of the error robustness techniques applied in this paper. Section III describes the design of the practical experience and the fourth, novel error robustness technique. Sections IV and V present the results and discuss the main observations. Section VI presents the conclusions.

2693-9177/22/$31.00 ©2022 IEEE
DOI 10.1109/QRS57517.2022.00092

II. Background and Related Work

In this paper we consider techniques for improving the intrinsic error robustness of neural networks (without considering traditional fault tolerance techniques such as modular redundancy). It is commonly held that CNNs have intrinsic error-masking ability [4], [9]. This is attributed to CNNs distributed structure and over-provisioning [11]. In critical occasions, however, errors can surpass a network's error-masking ability and lead to failure resulting in misclassification [12].

Research has been conducted on the fault tolerance characteristics of CNNs [13], [14]. One of the most relevant characteristics is the topology of the network. Emphasis is given to the type of layer that is used. Normalisation layers [15], for instance, are a type of layer that is used to increase the generalization accuracy of the network by bounding the output of specific layers to a predetermined range, preventing large changes. Due to this, Local Response Normalisation (LRN) layers also increase the error robustness of a network.

There are two generally accepted types of fault tolerance [9]. The first is *active* fault tolerance. Systems that employ this type have two components, one for detecting faults and another for controlling them. The main idea is that these systems detect faults as they appear and handle the effects using mechanisms that compensate for those effects. This compensation is achieved by taking the tasks that were being carried out by the faulty components of the neural network and assigning them to non-defective ones. The second type is *passive* fault tolerance. In contrast to the previous type, systems that use passive fault tolerance do not directly detect and manage faults. Instead, these systems make architectural changes to the neural network that increase its redundancy, thus indirectly compensating for the effects of faults.

Constraining the weights of CNNs improves their error robustness. This can be achieved through binding the magnitude of the weights in a layer to a predetermined range [16]. One technique using this idea is *Ranger* [10]. One possible consequence of soft errors (due to HW transient faults) is a large deviation in a neuron's output value. These errors may propagate through the network and affect the output. *Ranger* restricts the output values of neurons at certain layers in the network, thus clipping errors that occur before those layers.

A technique that draws some similarity to Ranger was named clipped activation [17]. Based upon empirical observations of the effects of hardware faults affecting intermediate layers of neural networks, the authors propose to clip the activation function by bounding the output values of intermediate nodes, within cut-off ranges. Their contribution, similarly to Ranger, is to specify how to define the output range for each layer of the neural network, while precluding the need for the training dataset nor modifying the parameters of any given network, that is, purely by bounding the activation function.

Due to the relevance of the soft error issue in critical systems, some techniques have been recently proposed. Algorithm-based error detection [18] explores the usage of three different strategies based upon checksums to verify the reduced output of convolutions. Moreover, algorithm-based fault tolerance [19], [20] has been applied to neural networks and shown to improve error robustness.

Full hardware redundancy is a very effective technique for protecting safety-critical systems [21]. However, the cost is too high for many applications and imposes additional weight and energy consumption. Hence, in this paper we compare network-level redundancy attained by increasing the number of neurons at each layer and allow the network to make free use of that available redundancy.

Authors have investigated the possibility of reducing memory voltage as a means to save energy while preserving accuracy [22] by injecting bit errors during training. It is known that reducing hardware voltage leads to effects similar to those of soft errors. Hence, authors aiming for voltage reduction could also benefit from error-resilient neural networks.

Dropout [8] is a regularization method that provides a simple and computationally efficient way of preventing overfitting. During each training iteration, each neuron has a predetermined probability p of being omitted. In the original version of dropout, both input and hidden layer neurons can be omitted and the probability p can be different between layers. Knowing that Dropout can prevent overfitting, in this paper we examine its ability to improve error robustness and, as the results show, that ability is at best very limited.

III. Methodology and Approach

The overall goal of the paper is to evaluate four techniques regarding their ability to improve the robustness of CNNs, namely redundancy [9], ranger [10], dropout [8] and a novel improved version of dropout called stimulated training. The first three techniques are described in the preceding section, as they belong to the state of the art, whereas the stimulated training technique is new and it is presented in this section.

Soft errors are a dependability threat originated by hardware faults [3], which represents a major concern for safety-critical systems that run CNNs over hardware accelerators. Such soft errors may cause SDCs that potentially lead to severe failures.

Fault injection campaigns were carried out, in which soft errors were inserted during the execution of CNNs, to empirically evaluate the accuracy of the different CNN models in the presence of soft errors. Such an experiment is representative of a scenario in which a CNNs executes directly on unreliable hardware, subject to soft errors.

A. Stimulated training

Stimulated Training is a novel adaptation of the Dropout technique developed in the scope of the present paper as a means to improve upon it. This technique is similar to dropout with one important distinction: instead of being omitted, neurons have their output values modified through random bit-flips on the output value. This is a crucial distinction, because the idea is for the neural network to automatically learn how to handle hardware faults during training.

The internal values of CNNs can have one of several data formats. The most common is the single-precision floating

point format with 32 bits from the IEEE 754-2019 standard [23]. This format can be divided into three parts: the sign bit, the exponent which has eight bits and the significand which has 23 stored bits.

The bit-flip operation, during training, chooses a random bit to be flipped. This manipulation mirrors the random bit-flip fault model mentioned above. The impact of each bit-flip depends on the original number and on the bit that is flipped. Resulting values can be vastly larger or smaller than the original ones, due to the floating point format.

In our experiments, the sign bit is not flipped during stimulated training (although it might during testing). Initial calibration experiments showed that the accuracy of a network is negatively affected by targeting that bit. Hence, without loss of generality and because one of the main goals is to maximize accuracy under the effects of hardware faults, the sign bit is excluded from stimulated training. This is a simple configuration based upon initial experimental results and including the sign bit once again would be perfectly feasible if it shows promising results.

The goal of this technique is to emulate soft errors during the training phase. By repeating this procedure, with a given probability across the targeted neurons, the neural network is trained to work in the presence of errors, thereby improving its robustness, so this is a modified learning technique.

The rationale behind the stimulated training technique is to let the neural network learn how to tolerate the hardware faults to which it is exposed during training. Much like the training dataset includes examples of images or entities to be classified by a neural network, the stimulated training technique provides examples of hardware faults and allows the network to develop internal fault tolerance. This is a fundamental distinction when comparing to the dropout technique, which discards random neurons to reduce overfitting. We observe that training the network under the presence of emulated hardware faults actually improves accuracy when compared to the original dropout technique.

B. Fault injection

Faults in CNNs can occur randomly [24]. To study their effect, one can inject faults in the neural network models in a variety of manners. In this work, fault injection was used to emulate transient hardware faults in the CPU through single bit-flips in CPU registers following an uniform random distribution. Random bit-flips in CPU registers is an ISA-level fault model that is widely used in other similar studies and has been shown to accurately emulate soft errors [25]–[27]. Thus, the usage of this fault model is expected to provide more representative results than if high-level error injection had been used. However, because we use a fault model for CPU faults, the neural network has to be executed on the CPU and not in HW accelerators. This point is considered and discussed in the analysis of the results.

For the experiments, we use a software-implemented fault injection framework called ucXception [6][1]. This framework injects faults by altering the value of the CPU registers of a userspace process during its execution. If this process (in our context, the model) uses these modified registers, then its internal state will be modified, possibly leading to misclassifications.

Faults were injected randomly in every bit of the general purpose registers (e.g., RIP, RAX), and in the x87 FPU registers. Additionally, faults were exclusively injected during the testing (i.e., actual use) of the model. The model classifies every image in the test dataset and at the end of each iteration the outcome can be one out of three possibilities: No Effect, Hang/Crash or SDC. This last failure mode (SDC) means that the injected fault caused a misclassification.

We evaluate the robustness of the models to hardware faults by measuring the relative frequency of misclassifications (SDCs) during the testing stage with every dataset. To do so, every model is executed in a golden run – without any faults injected – to establish the expected behaviour. Subsequently, all fault injection experiments are compared to the outcome of the golden run for the purpose of measuring classification accuracy under hardware faults.

It is worth to highlight that the preference for software-implemented fault injection comes with the need to perform fault injection is a standard Linux machine supporting the complete training-testing pipelines. Nevertheless, an alternative possibility would be to use hardware-emulated errors, potentially reducing the temporal and spatial intrusiveness of the fault injector and possibly improving the reachability of the technique to target more registers (which are not accessible at the ISA level).

C. Datasets

The evaluation was performed on four datasets, which are described next. *MNIST* [28] is a collection of handwritten digits, divided into 60000 training and 10000 testing samples. The images are grayscale, sized 28x28 pixels, and there are 10 classes which correspond to a digit from 0 to 9. The images have an *IDX* file format which was read using existing functions from the PyTorch framework.

Fashion-MNIST [29] is a dataset that consists of Zalando article images of clothing and shoes. It is more demanding to reach good accuracy compared to *MNIST*. It keeps the same image dimensions, classes, structure and file format as *MNIST*.

The *German Traffic Sign Recognition Benchmark (GTSRB)* [30] dataset contains photographs of traffic signals. It has 51839 images divided into 43 classes, with each image having 3 channels (red, green and blue). The dimensions of the images range from 15x15 to 250x250. Within each class, the images differ in the angle, brightness and other characteristics, in order to provide some variation and encompass plenty of possible real-life circumstances. We resized them using the OpenCV library to a standard 28x28 size.

The *CIFAR-10* [31] dataset is composed of a subset of the *80 million tiny images* dataset. There are 50000 training images and 10000 testing images, which are labelled as one of 10 classes. The classes range from animals (*e.g.*, bird, cat)

[1]The fault injector is available at https://github.com/ucx-code/ucXception

to means of transport (*e.g.*, airplane, truck). The images are 32x32 pixels with 3 channels (RGB).

D. Neural network architectures

LeNet or LeNet-5 [32] is a widely used CNN architecture. It is composed of convolutional, pooling and fully-connected layers. We implemented the original LeNet architecture for all datasets. It has a few differences depending on the dataset, namely: the input size; the number of input neurons of the first fully-connected layer (400 for *CIFAR10* and 256 for other datasets); the number of channels (one for *MNIST* and *Fashion-MNIST*, 3 for *GTSRB* and *CIFAR10*). For short, we identify LeNet as architecture (1) in all subsequent figures.

In order to evaluate the impact of redundancy on LeNet when combined with the other techniques, we added one convolutional layer (*conv3*) and increased the input and output dimensions of the convolutional layers, as well as the number of neurons in the fully-connected layers. We called the resulting network architecture (2), and used it in every dataset as a baseline in a similar way to architecture (1).

IV. Results

Neural network architectures (1) and (2) were trained and tested with the four previously mentioned datasets. To evaluate the performance of the models and the techniques, the following metrics were computed: classification accuracy; training time in seconds for each model; and the distribution of failure modes mainly focusing on the proportion of SDCs. Other failure modes do exist, such as crashes and hangs, and there is also a possibility for errors to be masked and have no effect. We focus on SDCs because that is the failure mode which affects accuracy and gauges the robustness of models in the presence of soft errors.

Results of 43 fault injection campaigns, totalling more than 600 hours, were collected. The number of campaigns corresponds to the combinations of models with datasets examined. Five injections per bit, at 64 bits per register, considering 27 registers, lead to 8 640 experiments per campaign. This resulted in a total of 371 520 faults injected.

A. Training and testing accuracy

The accuracy results for the different models, datasets and training/testing are presented in Figure 1. Accuracy is fundamental because one aims to improve error robustness while maintaining or improving correct classifications.

The first observation is that architecture (2) has better accuracy than architecture (1), which is to be expected because the latter is a relatively simple architecture while the former has more layers, as described earlier.

Applying the Ranger technique improves the accuracy in both architectures, including in cases where it is combined with Dropout. This contrasts with the findings in the original paper [10], which state that Ranger does not affect the accuracy in LeNet. The differing findings can be attributed to many experimental differences between the papers, such as using a distinct fault injection technique or programming language, which has previously been shown to have an effect on the results [33].

Another remark is that Dropout has an inconsistent impact on training and testing accuracy. In architecture (1), it decreases accuracy in all datasets, whereas in architecture (2) it has a varying influence, reducing in *MNIST* and *Fashion-MNIST* while increasing in *GTSRB* and *CIFAR10*.

Different Dropout probabilities impact architecture (2) in varying ways, such as decreasing the testing accuracy in the *MNIST* and *Fashion-MNIST* datasets, while increasing it in *GTSRB* and *CIFAR10*. An exception occurs in Dropout with probability of 0.8, which consistently provides the lowest training and testing accuracy. This could be expected, due to the high number of connections that are dropped out.

Stimulated Training either maintains the baseline accuracy or has a small negative effect, depending on the dataset. It does show an improvement in accuracy when compared to the Dropout technique, in every dataset with the exception of *Fashion-MNIST*. This is noteworthy given that Stimulated Training consists of a modification of the Dropout technique.

B. Training time

The training time for each model is presented in Figure 2. In these models the training used GPU acceleration. All results exclude the loading time. The Stimulated Training models were trained using the CPU, rather than GPU, because the used version of PyTorch was incompatible with the CUDA version of the graphics card.

Ranger results are not presented in these figures, because the technique is applied to a model after training – it consists of a one-time overhead to perform instrumentation [10].

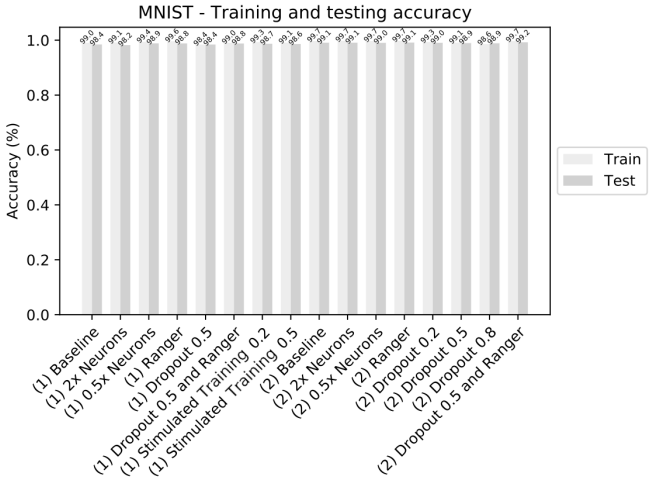
Generally we can conclude that architecture (2) has increased training time compared to (1) throughout every dataset. The training times for Stimulated Training were evaluated separately because the training was done using the CPU. We also trained baseline models for architectures (1) and (2) for the *MNIST* dataset for comparison purposes.

Training a Stimulated Training model of architecture (1) on the *MNIST* dataset results in a substantial increase of 31x in training time when compared to the baseline training. Training architecture (2) resulted in an increase of 52x in training time, which made it unfeasible to train models with that architecture for every dataset. This considerable increase is largely attributed to the usage of CPU for training, instead of the usual GPU training. Hence, we conjecture that a GPU implementation or some other optimization technique may lead to substantial improvements.

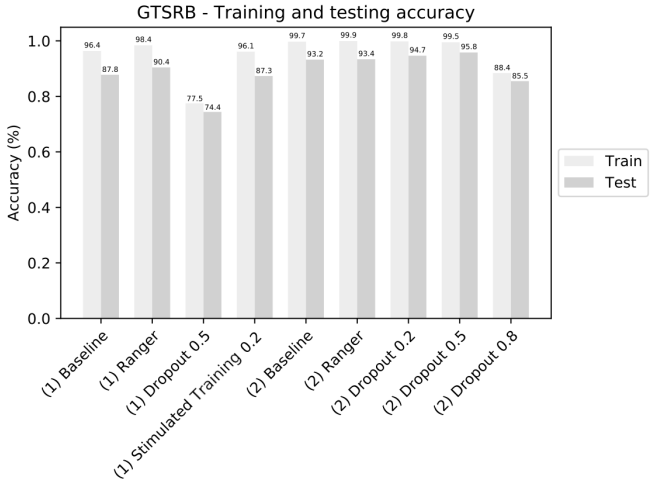
Stimulated training works by modifying the output values of neurons through random bit-flips. In principle, it is possible to emulate, or at least approximate, the effects of such bit-flips in a GPU. Hence, although our implementation is CPU-based, future work could optimize stimulated training to take advantage of GPU acceleration.

C. Failure mode distribution

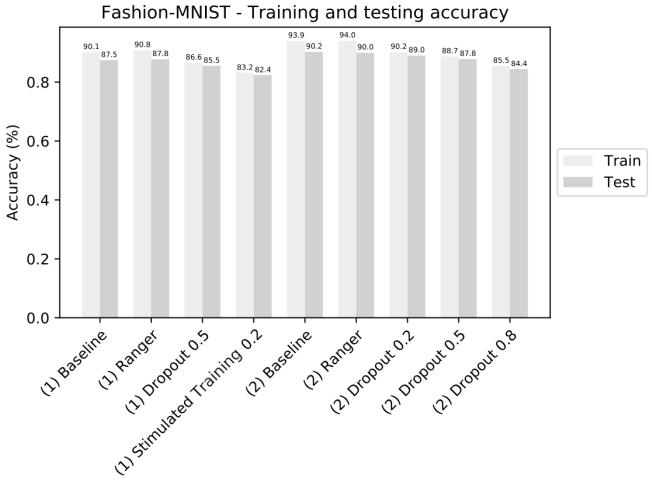
The most concerning failure mode arising after a bit-flip error is an SDC. Other less critical failures may occur and



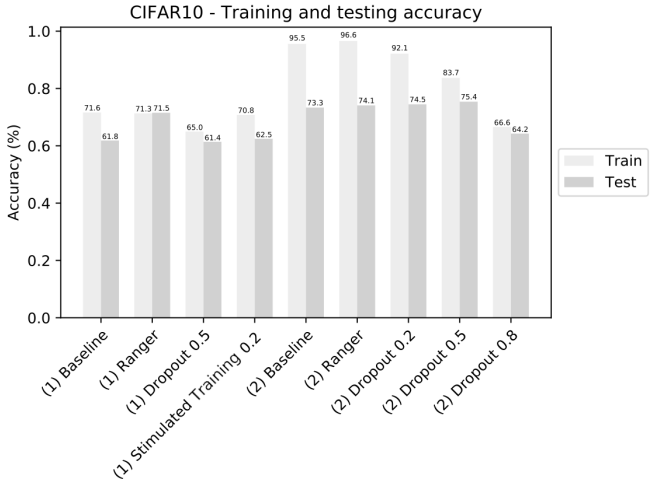
(a) Training and testing accuracy for the MNIST dataset



(b) Training and testing accuracy for the GTSRB dataset



(c) Accuracy for the Fashion-MNIST dataset



(d) Training and testing accuracy for the CIFAR10 dataset

Figure 1: Training and testing accuracy for each dataset

often faults injected are masked have no effect. This section presents the probability of an injected fault resulting in a silent misclassification, collected during the execution of the testing stage of every dataset. The results of the experiments for the *MNIST*, *GTSRB*, *Fashion-MNIST* and *CIFAR10* datasets are shown in Figures 3 through 6.

Figure 3 shows the results for models with Dropout for every dataset. Figure 4 presents the distribution of failure modes for models that utilise redundancy techniques for the *MNIST* dataset. Figure 5 presents the distribution of failure modes for models with Ranger for every dataset. The models were obtained by applying Ranger to baseline models of architectures (1) and (2) for every dataset. In the *MNIST* dataset, additional tests were made to evaluate the impact of Ranger on models with Dropout.

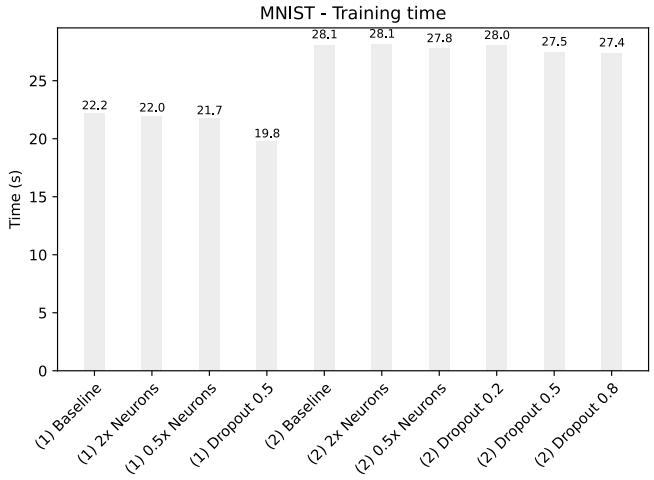
Figure 6 shows the distribution of failure modes for models with Stimulated training for every dataset. A Stimulated Training probability of 0.2 was used in every dataset, since this is the probability that provided the best results. Additionally, a probability of 0.5 was tested in the *MNIST* dataset in order to evaluate the difference in SDC rate. A detailed discussion of the main observations is presented in the section that follows.

Figure 7 provides a synthesis of the results by comparing the SDC probabilities across all techniques. For each technique, only the best instance is shown in the figure. For completeness, the figure contains also the probability of Crash/Hang failures and No Effect. The results in Figure 7 and some relevant observations are discussed in the section that follows.

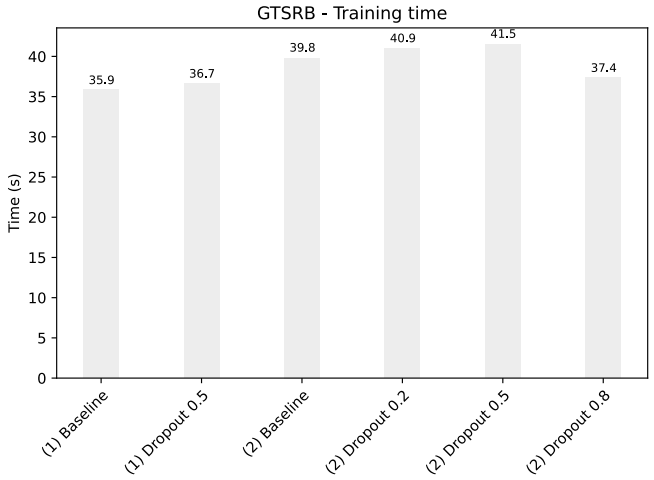
V. Discussion: robustness under hardware faults

Given that the SDC failure mode is the one that leads to misclassifications and reduces accuracy, the discussion of the robustness of the different techniques under hardware faults is focused on the probability of SDCs occurring.

When comparing the original LeNet architecture (1) with the modified architecture (2), one may observe a reduction

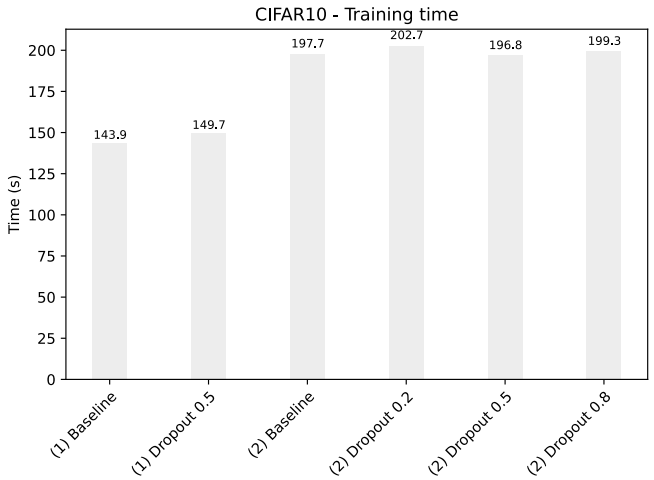


(a) Training time (seconds) for the MNIST dataset



(b) Training time (seconds) for the GTSRB dataset

(c) Training time (seconds) for Fashion-MNIST



(d) Training time (seconds) for the CIFAR10 dataset

Figure 2: Training time per dataset in seconds, using GPU acceleration (CUDA)

in SDCs in favour of architecture (2). Architecture (2) adds one convolutional layer and increases the input and output dimensions of the convolutional layers, as well as the number of neurons in the fully-connected layers. However, increasing the number of neurons *per se* does not improve the results – doubling or halving that number results in an increased SDC rate, with factors unchanged. The results in Figure 4 show little promise of benefiting from redundant neurons. Therefore, redundancy tends to reduce SDCs (and increase robustness) but that observation does not hold for redundant neurons.

Regarding the Dropout technique, it is observed to increase the SDC ratio in some cases and to have a mixed impact in some other cases. Specifically, Dropout is observed to increase SDC rates for both architectures, as shown in Figure 3a, it has varying results for example in Figure 3c, and it consistently decreases the SDC rate for example in Figure 3b. This leads to the observation that the Dropout technique, originally developed to reduce overfitting, does not improve hardware fault tolerance as one could speculate.

The Ranger technique [10] displays positive results, with some exceptions. The LeNet architecture does not benefit, in terms of SDC rate, from using the Ranger technique, or has varying impact. Architecture (2), which has redundancy, does in general benefit from the Ranger technique. For the *MNIST* dataset, Ranger significantly decreases the SDC rate of every tested model. In fact, applying Ranger to architecture (2) in the *MNIST* dataset resulted in the lowest SDC rate across all experiments. However, specifically in *CIFAR10*, applying Ranger increases the SDC rate between 3% and 22%. Hence, the Ranger technique is observed to provide good results, with some noteworthy exceptions, and also low training times.

We highlight that our observations are not directly comparable to the Ranger publication [10], because a different programming language, machine learning framework and fault injection tool are used [34]. These factors have been shown to influence the results of fault injection. Therefore, one

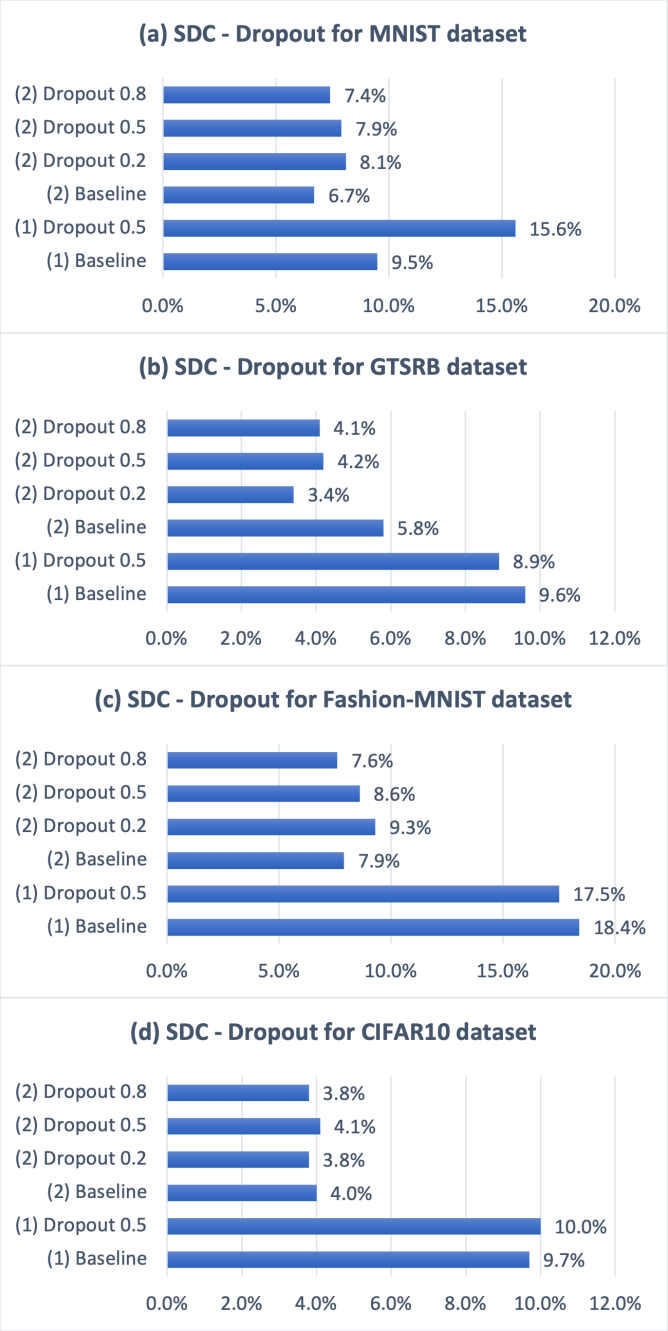


Figure 3: Distribution of failure modes for models using the Dropout technique

limitation of the results presented in this paper is that they only provide the means for internal comparison, but one may not extrapolate and compare with the original TensorFlow implementation and evaluation of the Ranger technique.

Regarding the stimulated training technique, we observe that applying it with a probability of 0.2 to architecture (1) decreases the SDC rate in every dataset. The reduction ranges from 39% in *MNIST* to 68% in *Fashion-MNIST*. In comparison, we observe that applying stimulated training to

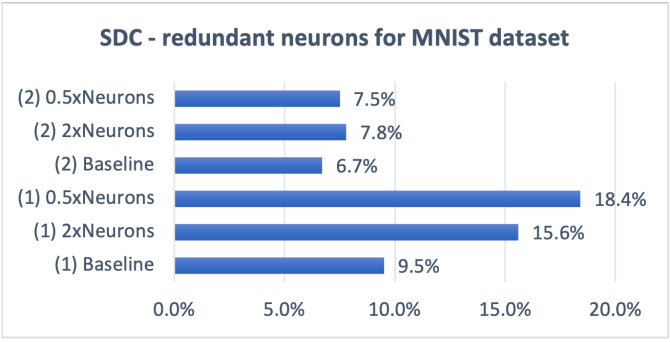


Figure 4: Distribution of failure modes for architectures (1) and (2) with redundant neurons for the *MNIST* dataset

GTSRB, *Fashion-MNIST* and *CIFAR10* results in the lowest SDC rates for these datasets. Using a probability of 0.5 actually increases the SDC rate. Hence, fine tuning this probability is an important step. Furthermore, training times for the stimulated training technique, using the CPU, are very high and therefore require future work to optimize it and take advantage of hardware acceleration.

VI. Conclusion

This paper proposes the technique of stimulated training, for improving the error robustness of convolutional neural networks, and compares its effectiveness with the techniques of dropout, ranger and redundancy. Four distinct datasets are used to evaluate the techniques through ISA-level software-implemented fault injection, with the results of 43 campaigns totaling 371 520 faults injected.

Overall, the ranger technique achieves the best results, closely followed by the stimulated training technique. Ranger combines low silent data corruptions with low instrumentation overhead and good accuracy. The stimulated training technique also achieves low silent data corruptions and high accuracy but the costly training time penalizes the technique. Nevertheless, no effort was made to optimize the training time, such as the commonly used GPU acceleration, and we thus conjecture that future work may lead to relevant improvements.

Stimulated training has an additional advantage over several other fault tolerance techniques, which is the absence of runtime overhead and instrumentation. The neural networks are trained to tolerate hardware errors along with the normal training process. The final network in general provides good accuracy and reduces the proportion of SDCs.

Redundancy also tends to reduce the proportion of SDCs, specifically when considering the addition of new layers. However, adding redundant neurons does not show any significant benefit and may in fact reduce error robustness, contrary to what one could intuitively expect.

Dropout, which is a regularization technique that prevents overfitting, is observed to have mixed impacts and in some cases worsening the proportion of SDCs. Hence, it may not be considered useful for improving the error robustness of neural networks.

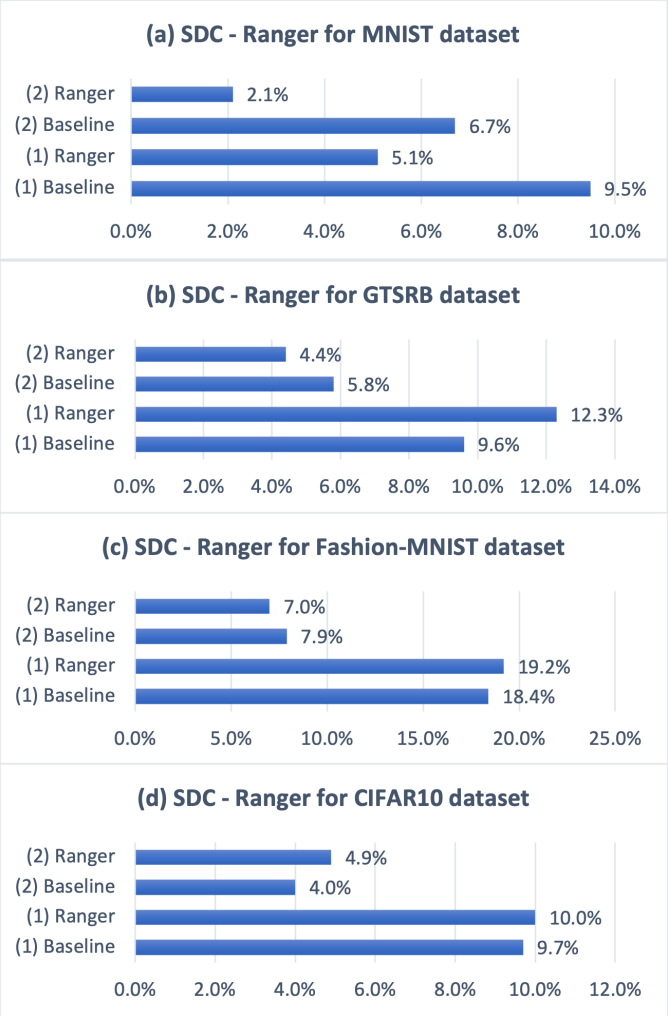


Figure 5: Distribution of failure modes for models using the Ranger technique

The results highlight the impact of fault injection technique and specific implementation details. Specifically, compared to previous experimental work, the ranger technique is found to provide a relevant contribution to the error robustness; however, the mean improvement is not as large as previously shown. Careful examination attributes the difference to two experimental details. First, we use PyTorch in C++ which is compiled into machine code, whereas previous evaluations of the ranger technique used Python implementations that are interpreted. Second, we used ISA-level fault injection, whereas previous evaluations used high-level error emulation. Hence, these results should caution practitioners to the fact that the results of fault injection are highly impacted by compilation/interpretation and by the fault injection technique.

ACKNOWLEDGMENT

This work was funded by the FCT - Foundation for Science and Technology, I.P., within the scope of project CISUC - UID/CEC/00326/2020 and by European Social Fund, through the Regional Operational Program Centro 2020.

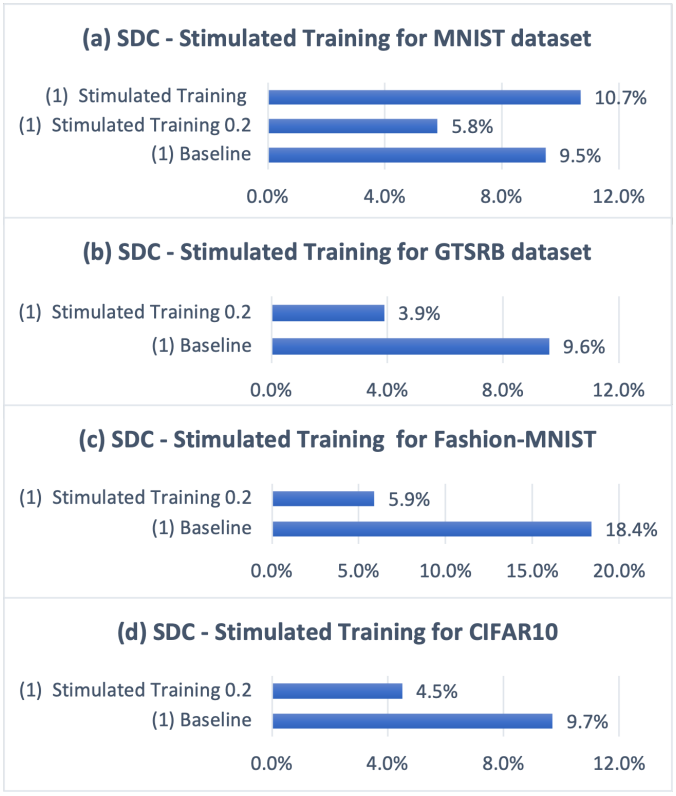


Figure 6: Distribution of failure modes for models with Stimulated Training

REFERENCES

[1] D. A. G. G. de Oliveira, L. L. Pilla, T. Santini, and P. Rech, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 791–804, 2015.

[2] N. Mahatme, S. Jagannathan, T. Loveless, L. Massengill, B. Bhuva, S.-J. Wen, and R. Wong, "Comparison of combinational and sequential error rates for a deep submicron process," *IEEE Transactions on Nuclear Science*, vol. 58, no. 6, pp. 2719–2725, 2011.

[3] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[4] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[5] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[6] P. D. Almeida, F. Cerveira, R. Barbosa, and H. Madeira, "ucXception: A framework for evaluating dependability of software systems," in *2022 IEEE 22th International Conference on Software Quality, Reliability and Security (QRS)*, 2022.

[7] F. Cerveira, R. Barbosa, H. Madeira, and F. Araújo, "The effects of soft errors and mitigation strategies for virtualization servers," *IEEE Transactions on Cloud Computing*, 2020.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[9] C. Torres-Huitzil and B. Girau, "Fault and Error Tolerance in Neural Networks: A Review," *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017.

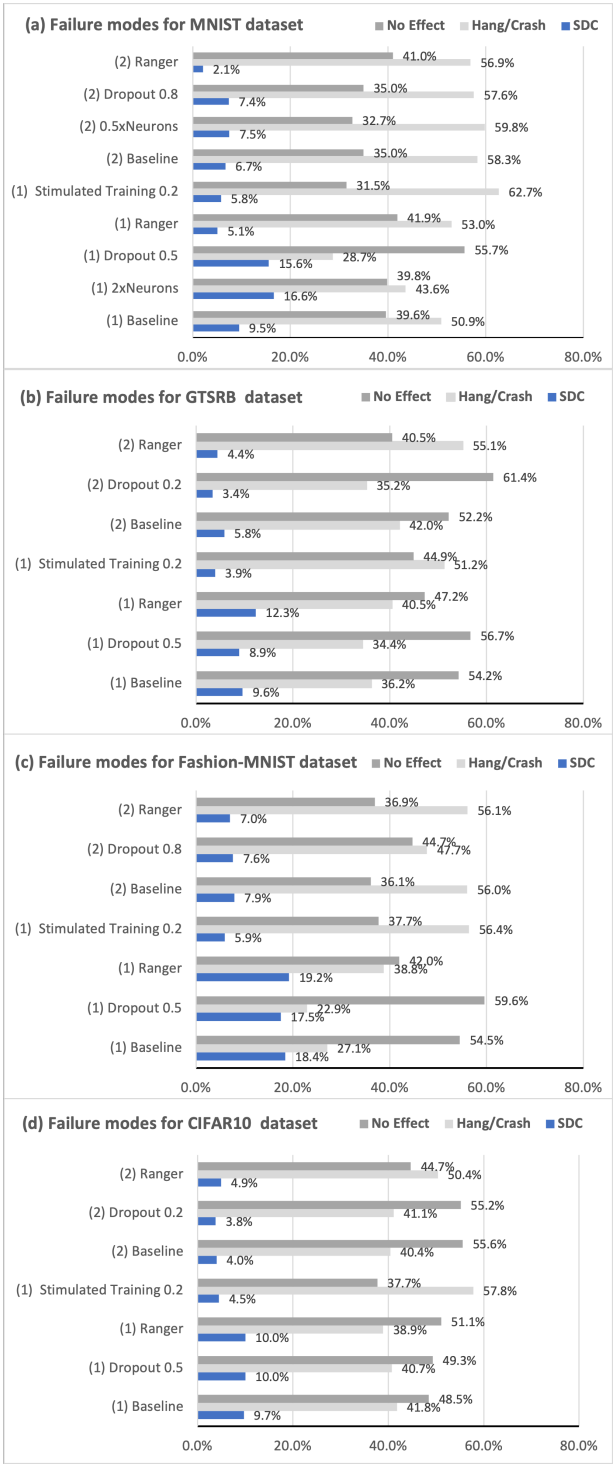


Figure 7: Comparison of failure mode probabilities for the best instance of each technique

[10] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2021, pp. 1–13.

[11] E. M. E. Mhamdi and R. Guerraoui, "When Neurons Fail," *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1028–1037, May 2017, arXiv: 1706.08884. [Online]. Available: http://arxiv.org/abs/1706.08884

[12] E. Ozen and A. Orailoglu, "Sanity-check: Boosting the reliability of safety-critical deep neural network applications," in *2019 IEEE 28th Asian Test Symposium (ATS)*. IEEE, 2019, pp. 7–75.

[13] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 1–12. [Online]. Available: https://doi.org/10.1145/3126908.3126964

[14] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "On the robustness of a neural network," *arXiv preprint arXiv:1707.08167*, 2017.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3065386

[16] N. Wei, S. Yang, and S. Tong, "A modified learning algorithm for improving the fault tolerance of BP networks," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1, Jun. 1996, pp. 247–252 vol.1.

[17] L.-H. Hoang, M. A. Hanif, and M. Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1241–1246.

[18] S. K. S. Hari, M. Sullivan, T. Tsai, and S. W. Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[19] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2018.

[20] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE transactions on computers*, vol. 100, no. 6, pp. 518–528, 1984.

[21] X. Iturbe, B. Venu, E. Ozer, and S. Das, "A triple core lock-step (tcls) arm® cortex®-r5 processor for safety-critical and ultra-reliable applications," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2016, pp. 246–249.

[22] L. Yang and B. Murmann, "Sram voltage scaling for energy-efficient convolutional neural networks," in *2017 18th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2017, pp. 7–12.

[23] IEEE, "754-2019 - ieee standard for floating-point arithmetic," https://ieeexplore.ieee.org/document/8766229.

[24] K. Pattabiraman, G. Li, and Z. Chen, "Error resilient machine learning for safety-critical systems: Position paper," in *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2020, pp. 1–4.

[25] H. Schirmeier and M. Breddemann, "Quantitative cross-layer evaluation of transient-fault injection techniques for algorithm comparison," in *2019 15th European Dependable Computing Conference (EDCC)*, 2019, pp. 15–22.

[26] B. Sangchoolie, K. Pattabiraman, and J. Karlsson, "One bit is (not) enough: An empirical study of the impact of single and multiple bit-flip errors," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 97–108.

[27] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. Leber, "Comparison of physical and software-implemented fault injection techniques," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1115–1133, 2003.

[28] A. Baldominos, Y. Saez, and P. Isasi, "A survey of handwritten character recognition with mnist and emnist," *Applied Sciences*, vol. 9, no. 15, p. 3169, 2019.

[29] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[30] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The german traffic sign recognition benchmark: a multi-class classification competition," in *The 2011 international joint conference on neural networks*. IEEE, 2011, pp. 1453–1460.

[31] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann, "Bit error tolerance of a cifar-10 binarized convolutional neural network

processor," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[32] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, conference Name: Proceedings of the IEEE.

[33] F. Cerveira, A. Fonseca, R. Barbosa, and H. Madeira, "Evaluating the inherent sensitivity of programming languages to soft errors," in *2018 14th European Dependable Computing Conference (EDCC)*. IEEE, 2018, pp. 65–72.

[34] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A flexible fault injection framework for tensorflow applications," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 426–435.