Selective Hardening of Critical Neurons in Deep Neural Networks

Selective Hardening of Critical Neurons in Deep Neural Networks

Annachiara Ruospo*, Gabriele Gavarini*, Ilaria Bragaglia*, Marcello Traiola†,
Alberto Bosio‡ and Ernesto Sanchez*
*Politecnico di Torino, DAUIN, Torino, Italy.
†Inria, University of Rennes, CNRS, IRISA, Rennes, France.
‡Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, France.

***Abstract*—In the literature, it is argued that Deep Neural Networks (DNNs) possess a certain degree of robustness mainly for two reasons: their distributed and parallel architecture, and their redundancy introduced due to over provisioning. Indeed, they are made, as a matter of fact, of more neurons with respect to the minimal number required to perform the computations. It means that they could withstand errors in a bounded number of neurons and continue to function properly. However, it is also known that different neurons in DNNs have divergent fault tolerance capabilities. Neurons that contribute the least to the final prediction accuracy are less sensitive to errors. Conversely, the neurons that contribute most are considered critical because errors within them could seriously compromise the correct functionality of the DNN. This paper presents a software methodology based on a Triple Modular Redundancy technique, which aims at improving the overall reliability of the DNN, by selectively protecting a reduced set of critical neurons. Our findings indicate that the robustness of the DNNs can be enhanced, clearly, at the cost of a larger memory footprint and a small increase in the total execution time. The trade-offs as well as the improvements are discussed in the work by exploiting two DNN architectures: ResNet and DenseNet trained and tested on CIFAR-10.**

***Index Terms*—Deep Neural Network, Reliability, Mitigation, Fault Injection**

I. Introduction

In the last decades, the growing complexity of emerging computing systems has called for enhanced computing paradigms. Among all the existing possibilities, artificial intelligence (AI) based solutions and, specifically, brain-inspired computing models like Artificial Neural Networks (ANNs), and in particular Deep Neural Networks (DNNs), have gained large interest in the industry and academia for their near-human computational capabilities [1]. DNNs are now considered attractive solutions for different areas including safety-critical applications such as self-driving cars, radars, flight control, robots, and space applications. Therefore, there is a compelling need for ensuring their reliability and tolerance to faults.

Being brain-inspired models, DNNs are traditionally considered intrinsically fault-tolerant and tightly robust. The first reason is related to their distributed and parallel structure, the second one to the redundancy introduced because of the over-provisioning [2]. Indeed, neural networks are made of a number of neurons higher than the minimal required to perform a computation. It means that they can tolerate some errors due to the excessive neurons budget [3]. This property implies that DNNs can tolerate a bounded number of potentially failing neurons without impacting the result of the computation; beyond that number, the precision degrades gracefully [4]. Nevertheless, when DNNs are deployed on resource-constrained hardware devices, single physical faults on hardware units might jeopardize the activity of multiple neurons, depending on how the DNN workload is scheduled on hardware resources [5]. The corruption of multiple neurons may lead to misprediction or, generally, unexpected outcomes. It is worth adding that not all neurons play the same role in a neural network: in the literature, it is claimed that individual network parts differ in their error resilience [6], [7]. Neurons within a neural network turn up having different fault tolerance and resilience levels. Some of them strongly contribute to the final task and their failures have a large influence on the degradation of the final predictions; others turn up being less important to the output accuracy (the concept is connected to the property of the over provisioning). Thanks to this, the latter are often removed from the network, for example through the pruning technique [8]. In the literature, the problem of assessing the resilience of artificial neurons and especially identifying the critical ones is addressed in many research projects. The proposals are different and are oriented towards two main directions: studying the contribution of each neuron to the global error during the training phase, e.g., [6], [9], [10], or their contribution to the final network prediction during the inference phase, e.g., [5], [11].

In this work, we present an approach to selectively protect a subset of neurons in deep neural networks by using a Triple Modular Redundancy technique. Firstly, we use one of the most recent methodologies to classify neurons based on their criticality, i.e., [5]. Then, to realize the selective hardening, we triple the computations and the parameters of a set of critical neurons. Hence, under the single fault assumption, faults are corrected thanks to the adoption of a majority voter. Clearly, this implies advantages and drawbacks: the reliability of the DNN is enhanced at the cost of increased memory footprint and Multiply–ACcumulate (MAC) operations. The aim of this work is to investigate (i) the existing trade-offs between memory footprint, execution time, and reliability, and (ii) the best percentage (*pNeu%*) of protected neurons required to achieve a substantial increase in the overall reliability without heavily impacting the neural network size and execution time.

The rest of the paper is organized as follows. Section II provides the reader with background knowledge on critical neurons and

978-1-6654-9431-1/22/$31.00 ©2022 IEEE

presents the related studies. Section III describes the proposed approach and Section IV outlines the case study. Next, Section V reports on the experimental results. Finally, Section VI draws conclusions and future directions.

II. Background

The intent of this section is to provide a basic background knowledge in the field. First, an overview of the existing fault models in artificial neurons is provided. Then, some of the most relevant related works in the literature are discussed.

A. Fault Models in Artificial Neurons

In the neural network field, each artificial neuron is considered as a single entity which can fail independently of the failure of any other [4]. Neural networks are viewed as distributed systems consisting of two principal components: neurons and synapses, i.e., the communication channels connecting the neurons. Faults in neurons or synapse are therefore independent of those of others. Specifically, it is worth to point out that, as neuron, we refer to each pixel in the output feature maps of convolutional layers and to each node in the pooling (min, max, average) or fully connected layers. Normally, batch normalization and activation functions (e.g., rectified linear unit, sigmoid, Gaussian) are not considered as independent layers, and thus they do not provide the network with additional neurons. In the literature, two fault models cover the most common faults in neurons:

- **Crash**: Neurons completely stop their activity. A crashed neuron is modelled by purposely setting its output to zero. This can also be known as dropout fault model [11].
- **Byzantine**: Neurons keep their activity but produce arbitrary values, within their bounded transmission capacity [4], [12].

It is worth underlining that a fault affecting a single neuron may not lead to a failure. This is not only due to the intrinsic definition of a fault [13], but specifically for the ANN property of being over-provisioned.

B. Related Works

Over the past decades, the understanding of the reliability, fault tolerance, and robustness of neural networks has gained increasing attention around the world [14], [15]. Among all the possible aspects, understanding the importance of individual neurons takes on great relevance when running deep and complex DNN models on systems with limited computing and memory resources. To mitigate the problems related to the size of DNNs, in the literature, many researchers provided pruning techniques to remove either redundant neurons or connections from over-parameterized neural models. However, we advocate that, to properly remove neurons or weights, a proper resiliency analysis is needed. In fact, as mentioned before, neural network parts differ in fault tolerance capability.

Spyrou *et al.* in [16] propose a neuron fault tolerance strategy for Spiking Neural Networks (SNNs). They propose a fault tolerance mechanisms based on dropout to nullify the effect of certain faults (by temporarily removing neurons during training with some probability) and some active fault tolerance techniques to detect and recover from the remaining faults. In [6], [9], the problem of identifying critical neurons in neural networks is addressed by using backpropagation of error gradients to find out those neurons that impact the output quality the least. The overall idea is to approximate those neurons that contribute the least to the global error with energy-efficient ones. The algorithms adopt the training set to forward and backward propagate values and then assign resilience scores based on the magnitude of their average error contribution. To the same end, Wang *et al.* in [10] address the problem of the voltage scaling technique applied to neural networks. This method is widely used for energy saving, but it may cause significant reliability challenges. They propose to build voltage islands with the same voltage to neurons with similar fault tolerance capability. To identify critical neurons, they use the derivative of the cost function to represent the final output quality degradation caused by the error of the neuron's calculation.

Afterwards, authors in [11] propose a method that differs from the above-mentioned gradient-based approaches, i.e., [6], [9], [10]. They characterize the resilience of individual neurons based on their average contribution to the output of the DNN. In other words, they state that neurons which strongly contribute to the output of a neural network have a greater impact on the classification accuracy degradation in case their output is disturbed. Their method bases on a Taylor decomposition and layer-wise relevance propagation (LRP) algorithm to compute the value for each neuron and shows to be more effective than the gradient-based ones.

Based on the same observation (the average contribution to the output of a neural network), authors in [5] recently proposed a similar approach to assign resilience scores to neurons while adding a key contribution. Unlike existing works in the literature which can be considered as network-oriented, they claim that in a multi-output neural network, the contribution of a single neuron can have a significant importance for individual classes and not only for the entire neural networks. As a matter of fact, neuroscience theories state that neurons become active in a particular pattern of neuronal activity based on precise input stimuli [17]. Therefore, they propose a class-oriented analysis and an algorithm to sort neurons based on both their contribution to individual classes and to the entire neural network.

III. Proposed Approach

This work presents a methodology to improve the reliability of DNNs by selectively hardening subsets of neurons based on their criticality. It leverages a Triple Modular Redundancy (TMR) mechanism to protect the functionality of the most important computations and prevent possible errors in them from propagating to the DNN output.

The approach builds on a preliminary phase: the ranking of neurons according to their criticality and resilience values. Among all the mentioned techniques (Section II-B), our interest landed on the methodology proposed in [5], where a traditional network-oriented analysis (neurons only considered as entities of the entire neural network) is combined with a class-oriented one (every neuron is likewise regarded as an entity of each individual class). In the following, we briefly summarize the approach. Specifically, neurons are sorted based on the magnitude of their average contribution over the training set, as also suggested in [11]. The algorithm proposed in [5] follows this procedure: (i) class-oriented analysis and (ii) network-oriented analysis. In the former, for each instance in the validation dataset related to the specific output class, a forward propagation cycle is performed, and a score is

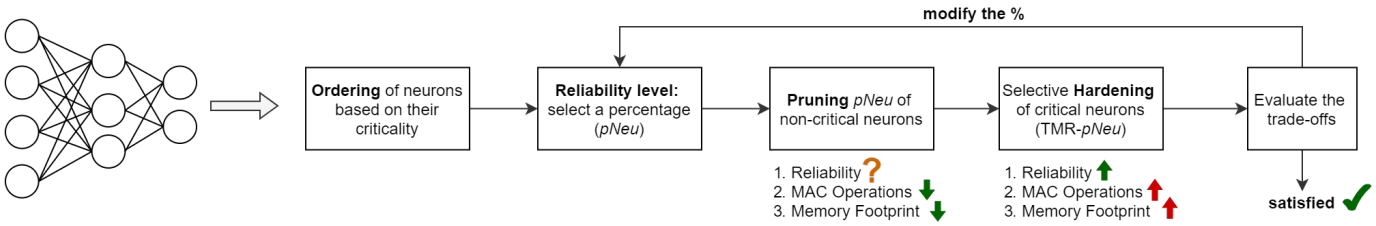


Fig. 1: Proposed Flow

given to each neuron by averaging the absolute output values produced during all inferences. At the end, neurons are sorted in descending order based on this resilience score producing a neurons criticality list. The process is repeated for the entire neural network, without differentiating between the output classes inside the dataset, and a single score map (list of neurons) is obtained. Finally, the network-oriented score map is updated based on the outcomes of the class-oriented analysis: with a given percentage (*pNeu*), neurons having a greater value in the class-oriented lists are updated in the final network criticality list. Notably, resilience scores are assigned to neurons considering both static and dynamic parameters: by averaging the output of the neuron, both the weights (*static* parameters) and the inputs (*dynamic* parameters) are taken.

In this paper, the proposed mitigation approach starts from the neurons criticality list, ordered based on the neurons criticality, and obtained as specified above. A diagram of the proposed flow is shown in Fig.1. First, as a measure of the desired level of reliability, a specific percentage of neurons (*pNeu*) must be selected to perform the following actions (detailed later):

1) Prune *non-critical* neurons, i.e., eliminate the selected *pNeu* from the bottom of the criticality list of neurons;
2) Reinforce *critical* neurons through a TMR-based approach where the *pNeu* is used to select a subset of critical neurons (TMR-*pNeu*) from the top of the list.

Although selectively hardening critical neurons allows increasing the reliability level of the DNN model, it comes at the cost of increasing the memory footprint (i.e., increasing weights and biases connected to TMR-*pNeu*) and also the number of MAC operations, which in turn increases the execution time. At the same time, the introduced computational overhead (in terms of MAC operations) is mitigated by pruning non-critical neurons. The reader should notice that, as the objective of this work is to improve the fault tolerance of a DNN model without increasing, as far as possible, its computational costs, the proposed solution leverages the removal of the same quantity of protected neurons.

This approach is based on the neuron output sparsity: a careful analysis of the list of critical neurons revealed the occurrence of neurons propagating, on average, zero values, that were located at the bottom of the list (non-critical neurons). Thus, the idea is to prune them to reduce the total number of MAC operations without affecting (or while slightly affecting) the DNN model accuracy. A similar approach is also proposed in [18], where the authors use a threshold to remove neuron activities below the threshold and therefore prune non-significant neurons. During this pruning phase, as shown in the diagram (Fig 1), not only the number of MAC operations is reduced, but also the memory footprint: indeed,

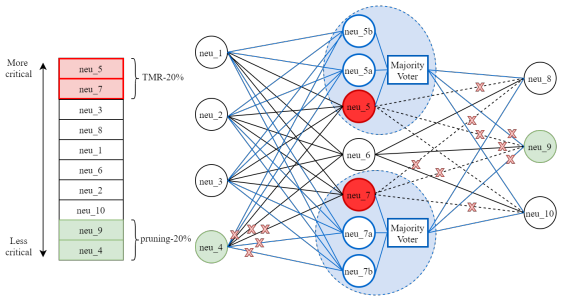


Fig. 2: The proposed methodology applied to a 3-layer feed-forward neural network: an illustrative example.

if neurons belonging to fully connected layers are removed, their weights and biases are deleted as well.

Afterwards, given the same initial percentage (*pNeu*), the selective hardening of critical neurons is applied. In more details, each neuron in the set (TMR-*pNeu*) is replaced with a TMR-based structure: not only its calculation is tripled, but also all the parameters related to it. Then, before transferring the value to subsequent neurons, a majority voter looks for the consistency of the three results. Basically, it ensures correctness, provided at least two out of three copies of the neuron remain operational and fault-free. Otherwise, if the values of the three copies do not match, the majority voter turns off the whole computation of the critical neuron, to avoid the propagation of wrong or, even worse, out-of-range values. As a result, the number of MAC operations and the memory footprint of the DNN application unavoidably might rise, but the reliability level of the DNN is improved.

For the sake of clarity, let us consider the simple 3-layer feedforward neural network illustrated in Fig. 2. In the example, *pNeu* is equal to 20%: a TMR-20 is applied, i.e., 20% of the critical neurons (at the top of the list) are protected and 20% of non-critical neurons (at the bottom of the list) is removed. The reader can note that: first, all the parameters (weights and biases) are tripled (the blue synapse), and second, the connections between the original critical neurons and the subsequent layers are cut and are replaced with those of the majority voter. Similarly, weights associated with non-critical neurons (if they are not shared - such as in fully connected layers), are removed as well.

To conclude, at the end of the proposed flow, if the designers are not satisfied with the resulting parameters (reliability level, MAC operations, and memory footprint), they can adjust the *pNeu* percentage to meet their goals.

TABLE I: DNNs Description

	Total Parameters [k]	Total Neurons [k]
ResNet-20	270	188
ResNet-32	464	303
DenseNet-121	6,956	578
DenseNet-161	26,483	926

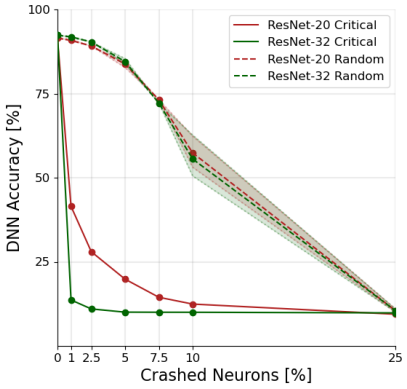
IV. Case Study

To prove the effectiveness of the proposed methodology, we used four different DNNs: ResNet-20, ResNet-32 [19], DenseNet-121, and DenseNet-161 [20]. The number of parameters and neurons of such DNN models are shown in Table I. All the DNNs were trained using CIFAR-10 [21], a well-known dataset used for the task of image classification. This dataset is composed of 60k 32x32 coloured instances, 50k training images and 10k test images. The DNN models are trained and tested with a 32-bit single-precision floating-point representation by using the PyTorch framework on a Linux server equipped with a dual Intel Xeon CPU E5-2680 v3 and 256 GB of RAM. To demonstrate the validity of the methodology, fault injection (FI) campaigns are performed. We resorted to the *crashed neurons* fault model: we model faulty neurons as neurons that completely stop their activity and the propagation of values. In the literature, this is also known as the dropout fault model. In practice, this error is introduced in the DNN by forcing to zero the output of the targeted faulty neuron. In the last experiment, we resorted to the *Byzantine neurons* fault model to model neurons that propagate random values. To run such FI campaigns, a specific library was built based on PyTorch to allow the injection of single and multiple faults on neurons. In more details, the fault injection is implemented by applying a binary mask to the output feature map of each layer at inference time. Each entry of the mask represents the occurrence of faults on the corresponding element of the target feature map. The computations of the masks are done offline, before running the injection campaigns.

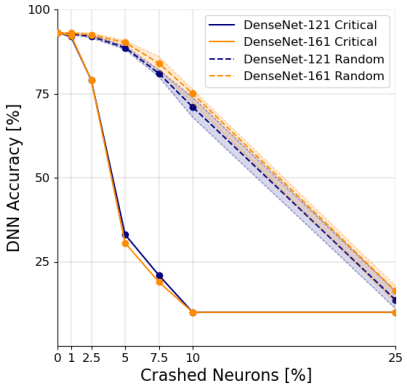
V. Experimental Results

To demonstrate the validity of the adopted ranking of neurons, we performed a first set of cumulative FI experiments where the same growing quantity of neurons was crashed (their outputs are cumulatively set to zero) in two scenarios. The reader should note that in this case the multiple fault assumption better reflects reality: in modern architectures, due to the size and the complexity of state-of-the-art neural networks, the computations of multiple neurons are assigned to a single hardware processing element. It means that a single physical fault affecting a processing element might compromise the activity of multiple neurons.

In the first scenario (*Critical*), the faulty neurons were selected from the top of the list of the critical ones; in the second (*Random*) they were randomly chosen from among all neurons. In both cases, the DNNs accuracy is reported in Fig. 3a and 3b for all the examined neural networks. The x-axis represents the increasing percentage of cumulative crashed neurons, whereas the y-axis shows the average accuracy that the DNN under assessment achieves by running all the instances of the CIFAR-10 Testset. As for the *Random* scenario, the FI experiments are repeated 10 times for each point of the x-axis (i.e., 1, 5, 10, 25 %): every time random neurons are selected (including the possibility of having



(a) ResNet-20 and ResNet-32



(b) DenseNet-121 and DenseNet-161

Fig. 3: Critical Neurons Vs. Random Neurons.

neurons in the set of critical). As it turns out, the accuracy of the DNNs falls dramatically if faults occur in critical neurons: this is where our mitigation proposal comes from. As stated, it leverages redundancy to (i) prune non-critical neurons and (ii) protect critical ones.

Therefore, to show that pruning does not affect, up to a certain point, the accuracy of the neural networks, we performed a second set of experiments. Fig. 4 shows how DNNs accuracy varies if a certain percentage of non-critical neurons is eliminated from the computations. It emerges that, in line with the redundancy property of DNNs, the deeper the DNN model, the higher is the quantity of non-critical neurons that you can remove without affecting the prediction accuracy of the network. For example, in DenseNet-161 it is possible to remove up to 15% of non-critical neurons while keeping the same accuracy. On the other hand, in reduced-size neural networks, such as ResNet-20, this bound is reduced to 5%. The pruning of non-critical neurons allows to exploit the intrinsic redundancy property of neural networks and, at the same time, avoid an excessive increase in the number of computations: the proposed selective TMR repeats three times all the arithmetic operations of critical neurons. Based on the outcome of this experiment (Fig. 4), in the selective hardening phase *pNeu* was limited to 5 percent to ensure that, in all our case studies,

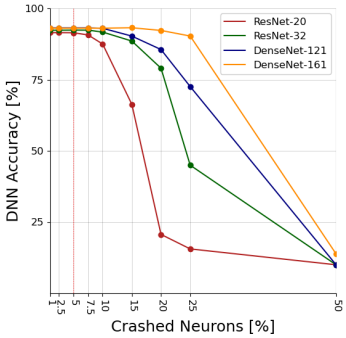


Fig. 4: Variation in DNNs accuracy due to the pruning of non-critical neurons.

pruning would not affect the accuracy and, thus, the behaviour of the DNNs.

In this light, in Table II we propose a complete analysis of costs: for every DNN and TMR-*pNeu*, both the variation of the memory footprint of the DNN application and the execution time (in terms of number of MAC operations) is reported. Specifically, the selective hardening methodology was applied with three levels of protection: TMR-1, TMR-2.5, and TMR-5. Data in Table II demonstrate that, thanks to the pruning of redundant neurons, the proposed methodology leads, on average, to a minimal increase in execution time (the maximum is reached with ResNet-20 and corresponds to +0.12%). It is interesting to point out that, in some scenarios, despite the unavoidable redundancy introduced by the TMR, the total number of MAC operations reduces. To explain this phenomenon, we resort to the formula used to compute the number of MACs per neuron (Eq. 1).

$$MAC_{\text{neu}} = C\_in * K\_h * K\_w \quad (1)$$

where *C_in* is the number of channels in the input feature map (its depth) while *K_h* and *K_w* the height and the width of the kernel of weights, respectively. This number is intuitively proportional to the size of the input channel and the kernel. In our DNN models, all the kernels have the same size (3x3). It means that Eq. 1 becomes only proportional to the first parameter (*C_in*). We have a reduction of MAC operations (Table II) if the TMR-*pNeu* is applied to neurons belonging to layers with a small *C_in*. As an example, most of the MAC operations added by the TMR in DenseNet-161 are due to critical neurons in the first layer: up to TMR-5, only in the first one where *C_in* is equal to 3. In deeper layers of DenseNet-161, *C_in* reaches a depth of 2,064 channels. In contrast, neurons removed by pruning are withdrawn from various layers (of different size, typically from deeper layers with deeper channels). This justifies why, despite the redundancy introduced by the proposed technique, the execution time might decrease. The same applies to the other DNNs under study. However, critical neurons in ResNet-20 and ResNet-32 are principally concentrated in layers with a small *C_in*, but also spread in deeper ones. Similarly to DenseNet-121 and DenseNet-161, the pruning acts in different layers. It is worth to point out that the maximum *C_in* in ResNet-20 and ResNet-32 is at most 64 (against a 2,064 of DenseNet-161). As regards the memory footprint, Table II evidences a general increase for every DNN and every TMR-*pNeu*. As explained in Section III, to protect the computation of a specific *pNeu* of critical neurons, all their weights and, overall, parameters, are replicated. Typically, kernels' size in initial layers is lower compared to that of the last ones and the most critical neurons are mostly located in them.

Cumulative Crashed Neurons: After discussing costs and gains of the proposed technique (Table II), in a further set of experiments we show the improvements in terms of reliability and robustness. Fig. 5 illustrates the outcomes of FI experiments, reproducing the worst-case scenarios: when critical neurons are crashed and completely stop their transmission activity. In the graphs, the selective hardening is used with three values of *pNeu*: 1, 2.5 and 5%. Aiming at reproducing the worst-case scenarios, the injections are not performed randomly, but following the order of critical neurons. Clearly, the accuracy of the DNNs remains stable when the *same* percentage of protected neurons crash. After that point, the accuracy of the DNNs slightly and smoothly decrease, specifically when a TMR-5 is applied. It means that, on the critical list, subsequent unprotected neurons are not as critical as previous ones. DenseNet-161, for instance, thanks to the TMR-5 maintains a prediction accuracy greater than 80% up to 8% of crashed neurons (the top 5% is protected by the selective hardening), in the worst case. Clearly, both in TMR-1, TMR-2.5, and TMR-5, the pruning of non-critical neurons is applied together with the hardening.

Cumulative Byzantine Neurons: As discussed in Section II, faults in neurons can be also modelled by forcing random values at their output (*Byzantine fault model*). We performed a further experiment to illustrate the behaviour of the DNNs when (i) random faults affect the unprotected DNN and (ii) random faults affect the protected DNN with different levels of TMR. The entire range of values that can be represented on 32 bits was used to force Byzantine faults. Experimental results are shown in Fig. 6 for ResNet-32. The x-axis reports the number of cumulative Byzantine faults that are introduced in random neurons (the experiments are repeated 10 times); the y-axis depicts the average accuracy that ResNet-32 achieves in four cases: with its original shape (i.e., unprotected), and with three different TMRs. For each point of the x-axis, the entire CIFAR-10 testset is run. Even with a reduced set of cumulative faults, it is clear that the DNNs with protected neurons have an enhanced ability to tolerate faults.

VI. Conclusions

This paper describes a technique to improve the reliability of DNNs by protecting the correct functionality of critical neurons. It is done by exploiting a Triple Modular Redundancy (TMR) mechanism combined with the pruning of non-critical neurons. Indeed, this work builds on two important and well-known factors: the over-provisioning of DNNs (i.e., their redundancy), and the different fault tolerance capability of neurons inside DNNs. Experimental results show that, due to the redundancy property, the methodology is more effective the bigger the DNN model is. The proposed approach comes with advantages and disadvantages, but, overall, it shows that the reliability and the robustness of the neural network can be improved if the computation of a reduced percentage of critical neurons is protected by a software TMR.

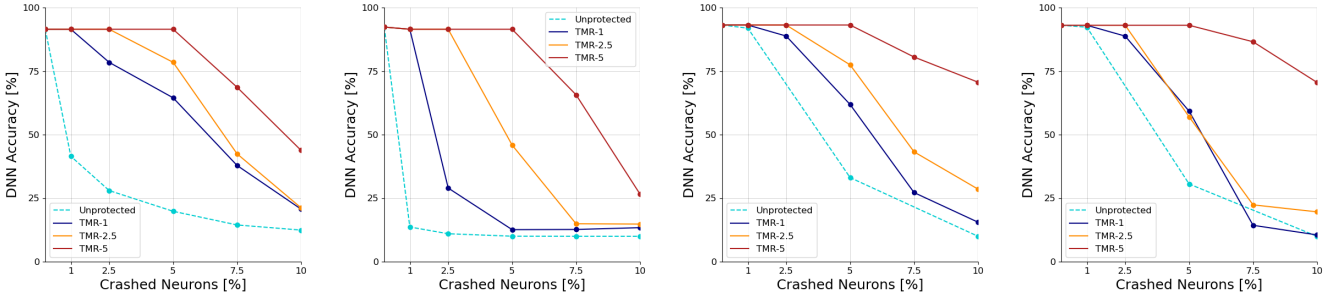
References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in

TABLE II: Overhead of the proposed technique.

TMR-%	ResNet-20		ResNet-32		DenseNet-121		DenseNet-161	
	Memory Footprint [kB]	MACs [M]	Memory Footprint [kB]	MACs [M]	Memory Footprint [kB]	MACs [M]	Memory Footprint [kB]	MACs [M]
TOTAL*	1,070	40.55	1,844	68.86	27,450	223.51	104,962	619.11
TMR-1	+7.67%	+0.04%	+1.66%	-0.01%	+0.03%	-0.16%	+0.01%	-0.20%
TMR-2.5	+10.06%	+0.05%	+7.13%	+0.03%	+0.03%	-0.37%	+0.01%	-0.42%
TMR-5	+22.56%	+0.12%	+15.45%	-0.08%	+0.04%	-0.87%	+0.01%	-0.76%

* Without the proposed techniques (pruning+hardening).



(a) ResNet-20 (b) ResNet-32 (c) DenseNet-121 (d) DenseNet-161

Fig. 5: Proposed Methodology with Crashed Neurons

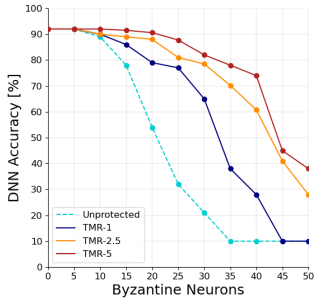


Fig. 6: Proposed Methodology with Byzantine Neurons on ResNet-32.

2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026–1034.

[2] S. Lawrence, C. Giles, and A. Tsoi, "What size neural network gives optimal generalization? convergence properties of backpropagation," 03 2001.

[3] S. S. Haykin, *Neural networks and learning machines*, 3rd ed. Pearson Education, 2009.

[4] E. M. El Mhamdi and R. Guerraoui, "When neurons fail," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 1028–1037.

[5] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on ai-oriented mpsocs," *Applied Sciences*, vol. 11, no. 14, 2021.

[6] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2014, pp. 27–32.

[7] X. He, W. Lu, G. Yan, and X. Zhang, "Joint design of training and hardware towards efficient and accuracy-scalable neural network inference," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 810–821, 2018.

[8] R. Yu *et al.*, "Nisp: Pruning networks using neuron importance score propagation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

[9] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 701–706.

[10] J. Wang *et al.*, "Enabling energy-efficient and reliable neural network via neuron-level voltage scaling," *IEEE Transactions on Computers*, vol. 69, no. 10, pp. 1460–1473, 2020.

[11] C. Schorn, A. Guntoro, and G. Ascheid, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 979–984.

[12] C. Allen and C. F. Stevens, "An evaluation of causes for unreliability of synaptic transmission," vol. 91, no. 22, pp. 10 380–10 383, 1994.

[13] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company, Incorporated, 2013.

[14] 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.

[15] A. Ruospo, E. Sanchez, M. Traiola, I. O'Connor, and A. Bosio, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocessors and Microsystems*, vol. 86, p. 104318, 2021.

[16] T. Spyrou *et al.*, "Neuron fault tolerance in spiking neural networks," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 743–748.

[17] "Memory systems of the brain: A brief history and current perspective," *Neurobiology of Learning and Memory*, vol. 82, no. 3, pp. 171–177, 2004, multiple Memory Systems.

[18] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 267–278. [Online]. Available: https://doi.org/10.1109/ISCA.2016.32

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[20] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.