Zero-Overhead Protection for CNN Weights

Stéphane Burel, Adrian Evans, Lorena Anghel

Zero-Overhead Protection for CNN Weights

Stéphane Burel[†], Adrian Evans[†], Lorena Anghel[‡]
[†]Université Grenoble Alpes, CEA, LIST, Grenoble, France
[‡]Université Grenoble Alpes, CEA, CNRS, Grenoble INP*, INAC-Spintec
Email: {stephane.burel,adrian.evans}@cea.fr, lorena.anghel@grenoble-inp.fr

***Abstract*—The numerical format used for representing weights and activations plays a key role in the computational efficiency and robustness of CNNs. Recently, a 16-bit floating point format called Brain-Float 16 (bf16) has been proposed and implemented in hardware accelerators. However, the robustness of accelerators implemented with this format has not yet been studied. In this paper, we perform a comparison of the robustness of state-of-the art CNNs implemented with 8-bit integer, Brain-Float 16 and 32-bit floating point formats. We also introduce an error detection and masking technique, called opportunistic parity (OP), which can detect and mask errors in the weights with zero storage overhead. With this technique, the robustness of floating point weights to bit-flips can be improved by up to three orders of magnitude.**

***Index Terms*—fault tolerance, error detection, quantized neural network, convolutional neural network**

I. Introduction

Convolutional Neural Networks (CNNs) are increasingly used in applications where safety requirements must be respected, notably in Autonomous Driving Systems. The implementation of these networks on hardware platforms must thus be tolerant of faults of any kind, including soft errors. Previous works have shown that in some cases, networks are tolerant of a large number of bit-flips, however, it has been shown that there are cases where a single bit-flip can cause a CNN's classification accuracy to drop dramatically [1]. We also note that by making networks robust to faults, we open the door to their implementation using advanced memory technologies which are intrinsically less reliable.

Improving the robustness of CNNs is an active field of study. Researchers have proposed hardware techniques, approaches based on identifying anomalies as well as techniques where the training process is modified. It is generally accepted that quantized neural networks are several orders of magnitude more robust than floating-point-based CNNs [2], [3]. The quantization process induces a loss of accuracy that may be problematic for some applications, therefore floating point formats remain important. In this paper, we propose *Opportunistic Parity* (OP), a zero-overhead technique to detect and mask errors in the weights of CNNs.

Other authors have shown that CNNs can tolerate a significant number of errors in the Least Significant Bits (LSBs) of their weights [4] which is also confirmed for the networks studied in this paper. Based on this observation, we propose a hardware-based technique to detect and mask errors in the weights of CNNs that employs *Opportunistic Parity* (OP). The idea behind the concept is to use some of the least significant bits in the weights in order to introduce a simple parity code to detect a single (or an odd number) of bit-flips. If the parity code detects an error, we replace the erroneous weights with zero and continue

In CNNs, it is known that masking an erroneous weight or activation value to zero has a minimal impact on the classification accuracy [5], [6]. Therefore, when a parity error is detected, by masking the data with zero, the impact of the error is minimized.

This technique was tested on three CNNs with different numerical formats. Our results demonstrate that the OP technique improves the robustness of floating-point-based CNNs by one to three orders of magnitude, with zero storage or area overhead.

This paper contains two main contributions. First, we perform a study of the robustness of three modern CNNs including their execution using the recent Brain-Float 16 (*bf16*) numerical format[1]. Secondly, we propose a new fault mitigation technique, *Opportunistic Parity*, that reduces the impact of bit errors in the weights. Tolerance to bit flips in the weights is important in high-reliability applications, where safety is a concern. Indeed, by making a CNN resilient to weights errors, we open the path to a wider acceptation of weight storage using advanced or emerging, lower-energy memory technologies, that are not very mature, being more prone to bit errors due to defects and variations.

II. Background

A. Convolutional Neural Networks

An artificial neural network is a machine learning model consisting of neurons, synapses and activation functions. The neurons are organized as layers and receive their input, or activations, from previous layers. The connections, or synapses, determine how activations propagate between layers. The configuration of the weights determines the behaviour of the network and these values are obtained during the training process. In this paper, our focus is on CNNs trained using supervised learning, as this is the type of network that is most widely used in computer vision.

Three networks are considered in our study : ResNet50 [7], SqueezeNetV1 [8] and MobileNetV2 [9]. All three networks are tested with the 2012 ImageNet data-set. ResNet50 was chosen as an example of a high performance network. It achieves state-of-the-art performance using deep residual layers, which

[1] [4], studies the IEEE764 16-bit floating point format.

consist of shortcut connections between layers to address the problem of a vanishing gradient during training.

SqueezeNetV1 and MobileNetV2 were chosen as examples of optimized CNNs. SqueezeNet is fully convolutional and uses *fire* modules, which are composed of a squeeze layer with a 1x1 convolution followed by an expand layer with 3x3 convolutions. It achieves low latency and has the lowest number of weights of the three networks. MobileNetV2 is based on an inverted residual structure with residual connections between bottleneck layers.

B. *Optimizing Neural Networks*

The high accuracy of CNNs comes at the expense of large memory storage for the weights and high computational complexity. These factors render their application in embedded systems more challenging.

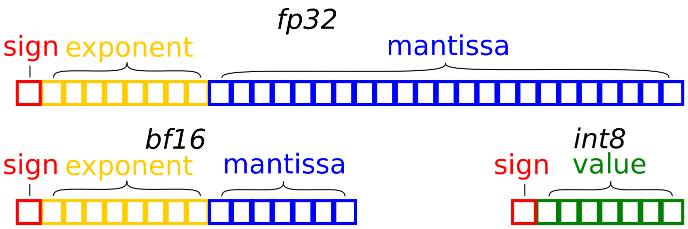


Fig. 1. Three Numerical Formats used for CNN Weight Values

In recent years, reducing the power consumption of CNNs, while maintaining accuracy, is an active field of research. Three main avenues are being explored. First, optimization of the network topology, which has resulted in the development of networks such as MobileNetV2 or SqueezeNet which reach state-of-the-art performance while reducing the number of operations and weights. Second, the development of dedicated hardware accelerators, such as Eyeriss [10] and ShiDianNao [11], which are optimized for the evaluation of neural networks. Third, the tuning of the numerical format, for example, the use of reduced precision floating-point or fixed-point reduces memory bandwidth and storage requirements. *bf16* is a recent 16-bit floating-point numeric format used in new Intel Xeon processors [12]. It has the same exponent width as a standard IEEE 754 single precision float. The length of the mantissa is simply reduced to 7 bits. When used in CNN accelerators, the multiply operations are performed using *bf16*, however, the sum is accumulated in a standard *fp32* format to limit the rounding error, as described in [13], [14]. These numerical formats are illustrated in Fig. 1.

C. *Characteristics of the CNNs*

For each of the three networks, we used publicly available ONNX models from [15]. The weights were converted to *bf16* format by simply rounding the mantissa to 7 bits. The conversion to *int-8* format was performed using the the N2D2 platform [16] using the methodology described in [17].

The classification accuracy is slightly lower when the numerical format for the weights is reduced. In Tab. I, we present the accuracy of the networks, in the absence of faults, as well the network requirements in terms of the number of multiply-accumulate (MAC) operations and the number of weights.

TABLE I
CHARACTERISTICS AND ACCURACY OF SELECTED NETWORKS FOR IMAGENET DATA-SET

Network	Numerical Format	Num. MACs	Num. Weights	Top-5 Accuracy
ResNet50	*fp32* *bf16* *int8*	3.9 G	25.5 M	91.90% 91.83% 88.08%
SqueezeNet	*fp32* *bf16* *int8*	352 M	1.2 M	80.35% 80.35% 78.74%
MobileNetV2	*fp32* *bf16* *int8*	300 M	3.4 M	89.91% 89.55% 86.39%

D. *Fault Model*

In this paper, we limit our study to bit-flips in the weights. For the purpose of this study, these bit flips could be the result of radiation-induced soft-errors, or they could be the result of timing errors or retention errors due to the weight storage memory being operated at an extremely low voltage, in order to reduce power [18].

III. OPPORTUNISTIC PARITY

Numerous, well known, techniques exist for protecting memories from bit-flips, including EDAC (Error Detection and Correction) codes with various detection and correction capabilities. In all standard approaches, additional check bits are added to the initial data, in order to provide protection or even error correction. Unlike these approaches, rather than adding additional check bits, we propose to directly modify the data, or the stored weights. In CNN applications, the least-significant bits of the weights is not critical [4]. Indeed, flipping a small number of these least significant bits (LSBs), leads to a very small loss in accuracy of the network.

In the simplest form, for each weight that initially had odd parity, we propose to flip the LSB in order to ensure that all the weights actually have even parity, as illustrated in Fig 2.

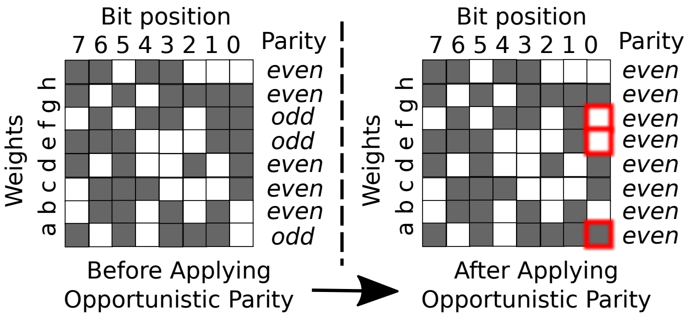


Fig. 2. Opportunistic Parity : Altering LSB to Obtain Even Parity

As presented, potentially, the LSB of every single weight could be flipped, which would have an impact on accuracy. However, we note that the storage word size of most modern memories is large (at least 64 to 256 bits). Multiple (N) weights are stored in one memory word. We can cover the entire memory word with a single parity bit, thus only the LSB of one of the N weight values needs to be flipped. This reduces the number of perturbations to the weights.

Different techniques could be used in order to select which of the N weight values to modify. One strategy would be to pick the weight with the largest absolute value, to minimize the relative error. Another strategy consists of picking the weight which is the least critical, but this would imply a costly analysis of the criticality of the weights. In the interests of simplicity, we arbitrarily pick the LSB of the memory word.

A. Fault Mitigation

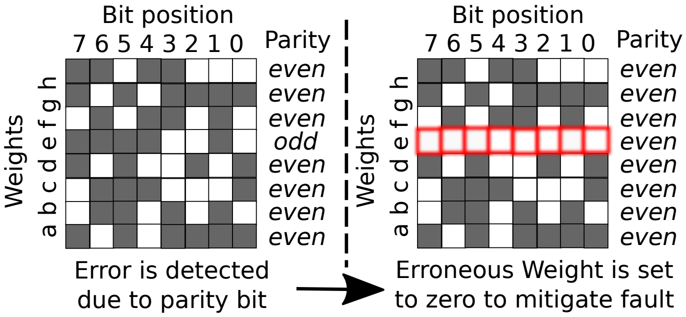


Fig. 3. Opportunistic Parity : Masking Detected Faults with Zero

Of course, a parity code only provides detection of an odd number of bit-flips. When a parity error is detected, the weight values can be replaced, or masked, with zeroes, as is illustrated in Fig. 3. Many authors [5], [6] have shown that in CNNs, masking erroneous values to zero is an effective fault mitigation technique. This approach prevents error scenarios where one of the Most Significant Bits (MSBs) gets flipped to one. These are the cases that most impact the accuracy.

B. Methodology

To analyze the efficiency of the fault mitigation strategy, we performed a series of fault injection campaigns. When performing fault injections on CNNs, the design of the experiment is very important since, even in the absence of faults, as seen in Tab. I, the classification accuracy of the network is not 100%.

Each of the data points we present is the average accuracy over N_{exp} = *20* fault injection experiments performed for a specific condition. A condition is defined as a tuple of the following variables : Network (ResNet50, MobileNetV2, Squeeznet), Numerical Format (*fp32*,*bf16*,*int8*), Bit Error Rate and Fault Mitigation Strategy (none, OP over individual weights, OP over 64 bit words, OP over 256 bit words). Each experiment consists of applying N_{batch_size} images and computing the accuracy for that batch.

The overall flow for the experiments is shown in Alg. 1.

1) Classification Methodology: When working with the ImageNet dataset, it is common practice to report the top-1 and top-5 accuracy. The former refers to the ability of the network to identfy the category of the image as the top ranked output. The latter, represents the ability of the network to place the correct category among the top five outputs. In the interests of simplicity, we have chosen to report only the top-5 accuracy.

```
for all Net in (ResNet50, MobileNetV2, SqueezeNet) do
  for all Numerical_format in (fp32, bf16, int8) do
    for all OP in (None, Weight, 64 bits, 256 bits) do
      for $BER$ from 1e-9 to 9e-1 do
        for i from 1 to $N_{exp}$ do
          $N_{faults} \leftarrow (num\_weights * BER)$
          $Faults \leftarrow random(weights, N_{faults})$
          Inject $Faults$ in weights
          for j from 1 to $N_{batchsize}$ do
            Classify $Image_j$
          end for
          Record average accuracy for batch
          Clear $Faults$ in weights
        end for
        Evaluate average accuracy for $N_{exp}$ batches
      end for
    end for
  end for
end for
```

Algorithm 1: Fault Injection Procedure

When reporting the impact of bit-flips on the weights, we have adopted the Bit Error Rate with Zero Accuracy Drop (BERZAD) metric proposed by Sabbagh [2]. BERZAD is defined as the maximum bit-error rate (number of erroneous weight bits divided by the total number of weight bits) such that there is zero loss of accuracy, based on a 95% confidence interval. A higher BERZAD value indicates that the network is more robust.

2) Batch Size: Due to the fact that the classification rate in the absence of faults is not 100%, the uncertainity in the fault free accuracy varies with the batch size. The batch size, (N_{batch_size}), is thus chosen such that the obtained error bars are less than 5%. In fact, it is a requirement for using the BERZAD metric, that the batch size be sufficiently large so that fault-free accuracy is never 5% below the nominal acuracy. In order to correctly choose the batch size, we performed a set of experiments without faults. The minimum and maximum accuracy was measured for 1000 batches of varying size as shown in Fig. 4. Although a batch size of 50 appears sufficient to ensure a variation of under 5%, we have chosen a larger batch size of 400. This was done, in order to ensure the uncertainty in the measured results, is well below 5%, for all three networks and numerical formats.

IV. Experimental Results

In this section we present the results of our fault injection experiments. First, we present an analysis of the sensitivity of the weights, per erroneous bit position. This first experiment is done to validate our approach of inverting the LSBs. Secondly we present the results of experiments where faults are injected in the weights, both with and without the OP technique.

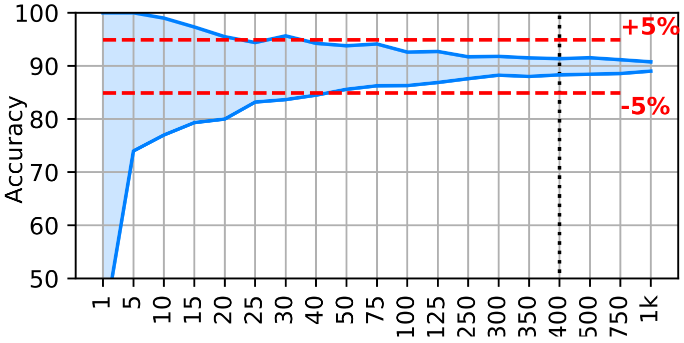
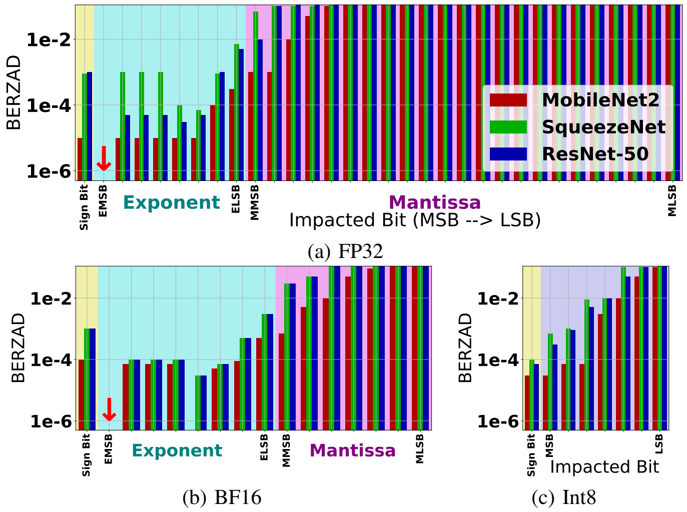


Fig. 4. Range of Accuracy for a given Batch Size for MobileNetV2 on *fp32*



(a) FP32

(b) BF16

(c) Int8

Fig. 5. BERZAD as a Function of Bit Position, for Three Numeric Formats and Three Networks

A. Analysis of Arruracy by Erroneous Bit Position

The sensitivity of the all the bit positions in the weights was studied for the three numerical formats (*fp32*, *bf16* and *int8*) for the three networks and the results are shown in Fig. 5.

First, we observe that that ResNet50 has a slightly higher overall BERZAD. This is not surprising, as MobileNetV2 and SqueezeNet are optimized, extremely compressed networks, and thus any perturbation is more likely to impact the results.

For both floating point numerical formats, there are cases where a single bit flip in the MSB of the exponent causes the accuracy to drop to zero. Thus, the BERZAD is zero, and hence off the graph (shown with a red arrow, due to the log scale). This is not the case for the *int8* quantized format, where a small number of bit flips in the MSBs does not produce an accuracy drop over 5%.

Consistent with previous studies [4], we see that exponent bits are more sensitive than mantissa bits. Also, for all the formats, including the *int8* format, the LSB can withstand a BERZAD of nearly 10e-1. This observation confirms our hypothesis and justifies the proposed the OP protecton technique.

The sensitivity of the *bf16* format is essentially the same as the 16 MSBs of the *fp32* format. We note that a *bf16* number is essentially a *fp32* with the 16 LSBs of the mantissa truncated (or preferably rounded). The *bf16* format, is thus more sensitive than the 16-bit IEEE 754 format, because the exponent field is larger.

B. Analysis of the Opportunistic Parity Technique

As discussed previously, OP can be applied based on different memory word sizes. We have considered three cases : (i) parity is adjusted for each weight individually, (ii) parity is adjusted for 64-bit words, (iii) parity is adjusted for 256-bit words. For comparison, we also present the classification accuracy in the absence of mitigation. These results are summarized in Tab. II.

TABLE II
CHARACTERISTICS AND ACCURACY OF OPPORTUNISTIC PARITY

Network	Num. Format	Level of Op.Par.	No-Fault Accuracy	BERZAD
MobileNet	*fp32*	None 32 64 256	89.9 89.9 89.9 89.9	1e-8 7e-6 5e-6 8e-7
	bf16	None 16 64 256	89.6 89.3 89.6 89.6	0 2e-6 5e-7 9e-7
	int8	None 8 64 256	86.4 79.0 85.8 86.1	2e-5 9e-6 7e-6 6e-8
SqueezeNet	*fp32*	None 32 64 256	80.3 80.6 80.5 80.3	9e-8 4e-5 4e-5 4e-6
	bf16	None 16 64 256	80.6 80.6 80.6 80.4	7e-8 4e-5 3e-5 2e-6
	int8	None 8 64 256	78.7 77.9 78.7 78.8	1e-4 2e-4 3e-5 8e-6
ResNet50	*fp32*	None 32 64 256	91.5 91.9 91.9 91.5	2e-9 8e-6 7e-6 2e-6
	bf16	None 16 64 256	91.7 91.8 91.7 91.8	0 7e-6 3e-6 4e-6
	int8	None 8 64 256	88.1 86.5 88.3 88.2	4e-5 1e-4 1e-5 6e-6

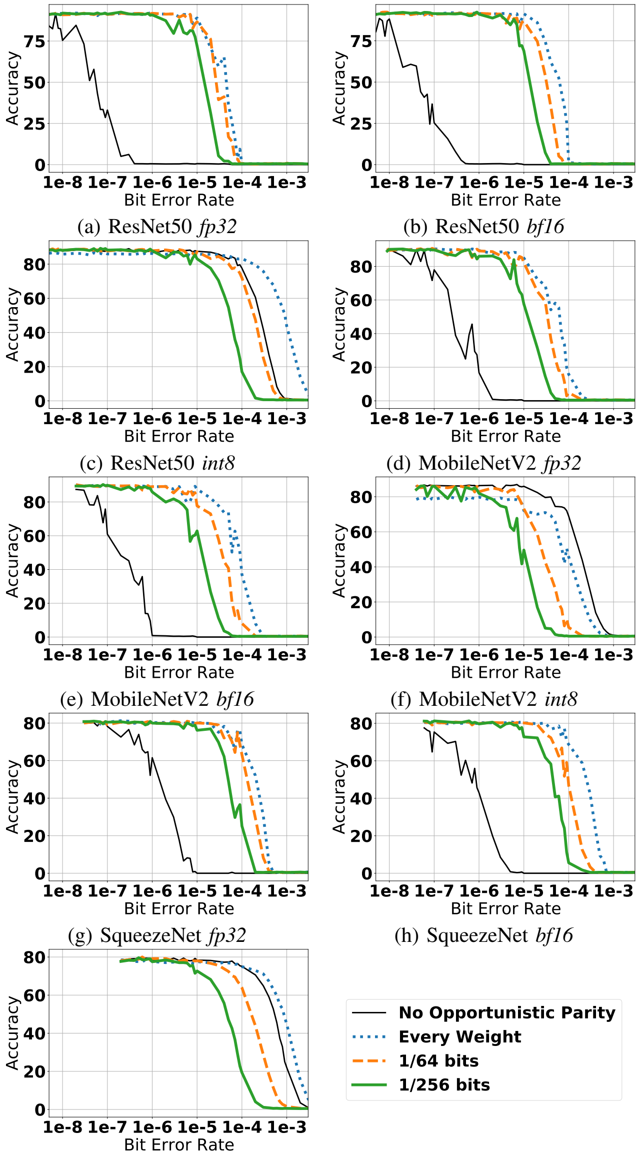
In all cases, with both floating point formats, OP greatly increases the BERZAD, indicating the network is more robust. For example, with MobileNetV2, with the *fp32* format, the BERZAD increases from 1e-8 to 7e-6, due to the OP at individual word level. Note that the impact of the weight modifications on the fault free classification accuracy is negligible.

For the case where the *int8* numeric format is used, when OP is applied at the 8-bit word granularity, there is a noticeable drop in fault-free accuracy (for example from 88.1% to 86.5% for ResNet50). However, it does provide a nearly 2.5x (4e-5 to 1e-4) increase in BERZAD. Indeed, when there is a parity error, only a single weight value is zeroed out, which has a minor impact on the accuracy. Still, for the same *int8* format, when OP is applied over groups of N=8 or N=32 weights (64, 256 bit words, respectively), there is virtually no drop in fault free accuracy, as very few bits are perturbed. The improvement in BERZAD is, however, reduced. In these cases, when there is a parity error, N weights get zeroed out and this has a real impact on the classification accuracy. Considering only the BERZAD metric, the benefits of OP appear to be limited for the *int8* format.

One shortcoming with the BERZAD metric, is that it provides no insight into the behaviour of the network, after the initial loss of accuracy. In Fig. 6, we plot the accuracy, as a function of the bit error rate. In these graphs, we see how the accuracy degrades with increasing bit error rate.

The black curves, show the behaviour in the absence of any protection, and we we see that for the floating point formats, the drop is quite sudden. This is due to the fact that, as soon as one, or a few, exponent bits are modified, the accuracy drops drastically. However, with OP, we prevent these extreme cases, and thus typically gain two orders of magnitude in BERZAD, before the accuracy drops off. In other words, using OP with floating point weights, it is possible to obtain the same fault tolerance (BERZAD) as a network using quantized weights stored in an *int8* format.

For the *int8* format, the results are mixed. For example, with ResNet50, Fig. 6c, with OP at the 8-bit word level (blue trace), we see that the accuracy starts to drop at a higher bit-error rate, and that the drop in accuracy is more gradual. For Squeezenet, Fig. 6i, OP at the word level, also provides a slight benefit. Unfortunately, for MobileNetV2, Fig. 6f, OP, even at the word level, degrades the fault tolerance. To summarize, the OP technique is highly effective at protecting weights stored in floating point formats. For 8-bit integer weights, in some cases, it can provide an improvement, but the results depend on the type of network.



(a) ResNet50 *fp32*

(b) ResNet50 *bf16*

(c) ResNet50 *int8*

(d) MobileNetV2 *fp32*

(e) MobileNetV2 *bf16*

(f) MobileNetV2 *int8*

(g) SqueezeNet *fp32*

(h) SqueezeNet *bf16*

(i) SqueezeNet *int8*

(j) Legend

Fig. 6. Impact of Opportunistic Parity on Accuracy

V. Related Works

Many previous works have studied the fault tolerance of CNNs. Sabbagh [2] introduced the BERZAD metric and performed an analysis of LeNet-5 and VGG-16, which are no longer state of the art networks. In this work, no fault-mitigation techniques are demonstrated.

Schorn [1] proposes a technique for modifying the weights in a CNN in order to increase fault tolerance. The technique is based on identifying the most critical weights and reducing their value in order to minimize the extent to which they propagate faults. After this change, a computationally expensive re-training is required to restore the original accuracy. The authors protected all the weights for the small LeNet-5 network. For the larger Squeezenet network, however, they only considered faults in a subset of the neurons. This technique is dependent on the network topology, requires costly re-training and it is not clear that it scales for complete, modern networks.

Kim [18] proposed a training method to improve robustness of CNNs to those bit-flips that are most likely to occur when an external SRAM holding the weights, is operated at low voltage. This technique requires a costly re-training of the network and it is not clear how it performs with recent, state-of-the-art networks.

Li [4] proposes a syndrome-based error detection technique, based on analyzing the statistics of the activation values. They achieve good fault detection for small networks. Draghetti [19] presents another detection method for soft errors in a video object detection application. The results for the current frame are compared with those from previous frames with the goal of identifying anomalies. With both of these works, the proposed technique introduces the problem of false positives (reporting of a fault, when no fault occurred).

Zhang [5] and Reagan [6] propose hardware techniques to mitigate computational errors. Both authors demonstrate that replacing erroneous values with zeroes is an effective way to mask faults. The OP approach builds on this observation.

Gambardella [20] studied the robustness of binarized networks. They propose selective redundancy for the most critical channels and also propose a fault aware scheduling technique, however, this work does not focus on memory errors.

Industrial providers of CNN accelerators [21] rely on more traditional Dual Core Lock Step (DCLS) to achieve error detection, although the area and power overhead is nearly 2x.

VI. Conclusion

In this paper, we have presented a fault injection study targetting the weights for three modern CNN architectures. Our results show that the modern, highly compressed networks, such as SqueezeNet and MobileNetV2 are more sensitive to bit-flips in the weights, than, for example ResNet50. We studied three numerical formats for the weights, including the new *bf16* format.

We have proposed a simple technique called opportunistic parity, which can be used to protect the memory used for weight storage for a CNN accelerator and requiring zero storage overhead. In the case of weights stored in a floating point format, including *bf16*, it provides a an improvement in bit error rate tolerance between 1-3 orders of magnitude with practically no loss in fault-free accuracy. For *int8* formats, the technique can provide approximately a 5x improvement in bit error rate tolerance, with a minor loss in fault free accuracy, however, the results depend on the type of network.

Our future work is focused on identifying new techniques to better protect quantized weights in highly compressed, modern networks.

References

[1] C. Schorn *et al.*, “An Efficient Bit-Flip Resilience Optimization Method for Deep Neural Networks.” IEEE, Mar. 2019, pp. 1507–1512. [Online]. Available: https://ieeexplore.ieee.org/document/8714885/

[2] M. Sabbagh *et al.*, “Evaluating fault resiliency of compressed deep neural networks,” in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019, pp. 1–7.

[3] F. F. dos Santos *et al.*, “Impact of Reduced Precision in the Reliability of Deep Neural Networks for Object Detection.” IEEE, May 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8791554/

[4] G. Li *et al.*, “Understanding error propagation in deep learning neural network (DNN) accelerators and applications.” ACM Press, 2017, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3126908.3126964

[5] J. J. Zhang *et al.*, “Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator,” in *2018 IEEE 36th VLSI Test Symposium (VTS)*. IEEE, 2018, pp. 1–6.

[6] B. Reagen *et al.*, “Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators.” IEEE, Jun. 2016, pp. 267–278. [Online]. Available: http://ieeexplore.ieee.org/document/7551399/

[7] K. He *et al.*, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/

[8] F. N. Iandola *et al.*, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and$<$ 0.5 MB model size,” *arXiv preprint arXiv:1602.07360*, 2016.

[9] A. G. Howard *et al.*, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017, arXiv:1704.04861.

[10] Y.-H. Chen *et al.*, “Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks.” IEEE, Jun. 2016, pp. 367–379. [Online]. Available: http://ieeexplore.ieee.org/document/7551407/

[11] Z. Du *et al.*, “ShiDianNao: shifting vision processing closer to the sensor.” ACM Press, 2015, pp. 92–104. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2749469.2750389

[12] “3rd generation Intel Xeon Scalable Processors.” [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/series/204098/3rd-generation-intel-xeon-scalable-processors.html

[13] D. Kalamkar *et al.*, “A study of bfloat16 for deep learning training,” 2019, arXiv:1905.12322.

[14] P. Micikevicius *et al.*, “Mixed precision training,” 2018, arXiv:1710.03740.

[15] “ONXX.” [Online]. Available: https://github.com/onnx/models

[16] O. Bichler, “N2D2,” https://github.com/CEA-LIST/N2D2.

[17] M. Nagel *et al.*, “Data-Free Quantization Through Weight Equalization and Bias Correction.” IEEE, Oct. 2019, pp. 1325–1334. [Online]. Available: https://ieeexplore.ieee.org/document/9008784/

[18] S. Kim *et al.*, “MATIC: Learning around errors for efficient low-voltage neural network accelerators.” IEEE, Mar. 2018, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/8341970/

[19] L. K. Draghetti *et al.*, “Detecting Errors in Convolutional Neural Networks Using Inter Frame Spatio-Temporal Correlation.” IEEE, Jul. 2019, pp. 310–315. [Online]. Available: https://ieeexplore.ieee.org/document/8854431/

[20] G. Gambardella *et al.*, “Efficient Error-Tolerant Quantized Neural Network Accelerators,” in *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2019, pp. 1–6.

[21] K. Matsubara *et al.*, “4.2 a 12nm autonomous-driving processor with 60.4tops, 13.8tops/w cnn executed by task-separated asil d control,” in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 56–58.