C-031

Analyzing the Impact of Weight Faults on Recognition Accuracy in Binary Neural Networks

Yukiya Miura† Riku Sawahata†

Abstract

Binarized neural networks have been proposed as a lightweight network for implementing neural networks in hardware. Although neural networks are known to be fault-tolerant, however, hardware is susceptible to physical faults and soft errors. In this paper, we assume that stuck-at faults cause the change in weights of neurons, and investigate the effect of these on the recognition accuracy of both standard neural networks and binarized neural networks. Experimental results show that the binarized neural network has a smaller decrease in recognition accuracy than a standard neural network. Therefore, the binarized neural network has an advantage over the standard neural network even if there are faults in the weights of neurons.

Keywords*: binary neural network, hardware implementation, stuck-at fault, weight fault*

1. Introduction

As computer performance improves, neural networks (NNs), which are networks of neurons connected in multiple layers, are widely used in various fields, such as image recognition including handwritten character recognition and object detection, and voice recognition [1]-[4]. In order to improve the recognition accuracy, a large-scale neural network consisting of a large number of neurons is required. However, a large-scale neural network requires a huge amount of calculation for learning and inference. Even with high-performance CPUs and GPUs, calculations take a long time, and it has been pointed out that it is not suitable for image recognition, which requires real-time responses [1]. In addition, the use of GPUs is also problematic due to their large power consumption [5]. Therefore, in order to improve processing speed, it is desirable to implement NNs in hardware, and there have been many reports on implementing NNs in hardware [6]-[9]. In particular, from the viewpoint of easy implementation and power consumption, hardware implementation of NNs using FPGAs has become mainstream [7], [9]. Typically, NNs perform calculations using real numbers, and NNs with high recognition accuracy require large amounts of data and calculations. Therefore, if an NN program is implemented as is in hardware, the hardware (FPGA) resources are insufficient, and as a result, hardware implementation of an NN is difficult. Therefore, it is necessary to reduce the amount of data and calculations and implement an NN with a small amount of hardware. Therefore, a binarized neural network (BNN) has been proposed, in which the weights, inputs, and outputs of activation functions are limited to two values (−1 and +1) [10]-[12]. The recognition accuracy of BNNs is almost the same as that of standard NNs which use real numbers. As the parameters in BNNs are limited to two values, they are suitable for hardware implementation. In addition to BNNs, NNs quantized to three values (−1, 0, +1) have also been proposed [13]. It is generally known that NNs are fault-tolerant, but most previous research has handled standard NNs using real numbers [14]-[19]. Implementing an NN in hardware means that various physical faults can affect the recognition accuracy of the NN. In particular, with BNNs, parameters such as weights are limited to two values, so there is concern about the impact of faults on accuracy. In this study, we investigated the effect on recognition accuracy when a fault changes the weights of the BNN, and compared whether the BNN or the standard NN is more effective against faults. Experimental results showed that the BNN is more resistant to weight change faults than the NN. The rest of this paper is organized as follows. Section 2 explains an overview of the NN and the BNN and their hardware implementation. In Section 3, the experimental contents, results, and discussions are presented. In Section 4, we discuss a comparison of the NN and the BNN with faults. Finally, in Section 5, the conclusions of this study are presented.

2. NN, BNN and Hardware Implementation

A neural network (NN) is a machine-learning model that mimics the mechanism of biological nerve cells [20]. The structure of the NN is shown in Fig. 1. The NN is roughly divided into an input layer, a hidden layer, and an output layer, and each layer is composed of multiple neurons. Each neuron is connected to multiple neurons in the previous layer by directed edges, and each connection has a weight. These weights control the influence of the input from the previous layer on the next neuron. After receiving the weighted input, each neuron performs a nonlinear transformation using an activation function, and the output becomes the input to the neuron in the next layer. Convolutional neural networks (CNNs), which are used for image recognition, use floating-point multiply-accumulation (MAC) operations, and CPUs and GPUs perform these operations with 8-bit or 16-bit precision, requiring a large amount of data and calculations [21].

† Faculty of Systems Design, Tokyo Metropolitan University

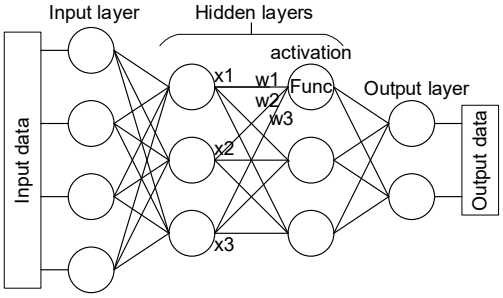


Fig. 1 Structure of NN

To calculate an NN in real-time, hardware implementation, especially implementation on an FPGA, is necessary. However, an NN that handles real numbers requires a large amount of data and calculations, and the hardware resources are insufficient as they are. Therefore, to reduce the amount of data and calculations, a binarized NN (BNN) has been proposed, which limits the input and output of the activation function to two values, +1 and −1 [10], [11]. This allows data (weights) to be expressed in one bit, and multiplication operations can be implemented with one XNOR gate. This significantly reduces the amount of resources required for hardware implementation. Note that (+1, −1) is encoded as (1, 0). It has also been shown that the BNN's recognition accuracy is comparable to that of standard NNs that use floating point numbers [10], [11]. Details of the BNN used in this study are explained in Section III.

On the other hand, when implementing NNs in hardware, there is concern about the impact of hardware faults on recognition accuracy. FPGAs used in hardware implementation of NNs consist of multiple LUTs interconnected via wires and switch matrices. SRAM-based LUTs store mask values (output values) in memory. Therefore, FPGAs may receive soft errors, which are bit inversions of memory values caused by energetic particles, and wire breaks or shorts [22]-[26]. Therefore, it is important to evaluate the impact of hardware faults on recognition accuracy. In particular, in BNNs, each neuron weight (+1, −1) is stored in the memory as a single bit, so there is a possibility of faults that change the weights due to soft errors, which are bit inversions, or memory access errors. Therefore, for BNNs, it is necessary to evaluate the impact of faults that cause weight changes on recognition accuracy.

3. Experiment, Results and Discussions

3.1 Experiment Contents

3.1.1 BNN and NN

In BNNs, weights and activation function outputs are expressed as binary values (+1 and −1). This means that when training the BNN, both real-valued weights and activations are constrained to +1 or −1. Two types of binarization functions are used to convert real-valued variables to these two values (Eqs. (1) and (2)). Eq. (1) is a deterministic binary function, and Eq. (2) is a stochastic binary function.

$$x^b = \begin{cases} +1 & \text{if } x \geq 0\,, \\ -1 & \text{otherwise}\,, \end{cases} \tag{1}$$

where x^b is a binarized variable (weight or activation) and x is a real-valued variable.

$$x^b = \begin{cases} +1 & \text{with probablity } p = \sigma(x), \\ -1 & \text{with probablity } 1 - p, \end{cases} \tag{2}$$

where σ is a hard sigmoid function (Eq. (3)).

$$\sigma(x) = \text{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right) \tag{3}$$

The BNN program used in this study is a PyTorch version of BinaryNet (BN) that is publicly available on GitHub [27]. When training the BN, the weights are binarized using the deterministic binarization function, and the activation function values are binarized using the stochastic binarization function.

To compare with a standard NN that handles real numbers, we trained a modified BN with the binarization function disabled and used it in the experiments.

3.1.2 Benchmark

MNIST (Mixed National Institute of Standards and Technology database) is used to verify the recognition accuracy of the faulty BN and NN. MNIST is a dataset of handwritten digit images. It consists of image data of handwritten digits from 0 to 9 and label data representing the correct answer to the digit written in the image [28]. There are 60,000 pairs of image data and label data for training and 10,000 for testing. Each image is expressed in grayscale of 28 × 28 pixels with brightness values ranging from 0 to 255. The recognition accuracy of the fault-free NN and BN are 97.53% and 97.06%, respectively.

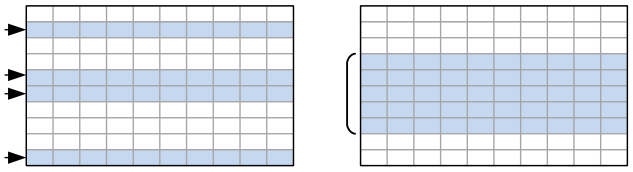
3.1.3 Fault Injection

In this study, it is assumed that the NN weights change due to a fault. The BN is composed of four layers, with each layer having a different weight size and being binarized or not. The weights are stored as a two-dimensional array, with the output of the first layer being (784 rows) × (6144 columns) of binarized weights, the output of the second and third layers being (6144 rows) × (6144 columns) of binarized weights, and the output of the fourth layer being (6144 rows) × (10 columns) of real-valued weights. Note that the input to the first layer is real-valued.

Since the weights in the BN are treated as array data, it is assumed that they are stored in memory when the BN is implemented in hardware. A bit inversion of the memory value caused by a soft error is one of the memory faults. However, soft errors generally affect the value of only a few bits. Due to the fault-tolerant nature of NNs, changes in the value of a few bits have a negligible impact on recognition accuracy. Our preliminary experiments also supported this finding. Other faults are assumed to be stuck-at faults in memory cells and peripheral circuits, such as decoders and

sense amplifiers. For this reason, the weight value caused by the stuck-at fault is assumed to be one of three values (+1, 0, −1) for the standard NN, and two values (+1 and −1) for the BN. In this study, the fault injection was simulated by rewriting the weights of each layer of the NN and BN after training. The purpose of this study is to investigate the effect on the recognition accuracy of the BN of changes in weights due to faults. Therefore, the number of affected weights (i.e. faults) is intentionally increased to achieve this purpose.

Two types of experiments were conducted for the weights of each layer: (Exp. 1) changing the weights based on their magnitude, and (Exp. 2) specifying rows or columns of the array and changing all weights within that range. Furthermore, in Exp. 2, two selection methods were applied: (Exp. 2a) randomly selecting individual rows or columns, and (Exp. 2b) randomly selecting consecutive rows or columns (Fig. 2). Exp. 2 was performed five times, and the impact on recognition accuracy was evaluated using the average value. Below, the weight value is denoted as W.



(a) Exp. 2a (b) Exp. 2b

Fig. 2 Weight change method

3.2 Experimental Results and Discussions

3.2.1 Experiment 1

Exp. 1 investigated the effect of the NN weight magnitude on accuracy. In a standard NN, the weights are real values between −1.0 and +1.0. Therefore, the weights, based on their magnitude, are changed in the following four ways to investigate the effect on accuracy: original values: $|W| \geq 0.8$ or $|W| \leq 0.2$ → faulty values: 0 or (+1 or −1). Here, the faulty values +1 and -1 are chosen to be the closest to the original value (e.g., original: W = +0.1 → fault: W = +1, and original: W = −0.1 → fault: W = −1).

In the BN, the weights in the same positions as those changed in the standard NN are changed. There are three settings for changing the weights: changing all selected weights to +1 or −1, or inverting the sign of the weights (+1 to −1 or −1 to +1). Note that the fourth layer of the BN has real-valued weights, so the weights in the fourth layer were changed in the same way as in the standard NN.

The effect of weight changes on accuracy is shown in Figs. 3 and 4. As shown in Fig. 3, changing large weights significantly affects recognition accuracy in the standard NN. A technique called pruning, which removes connections between neurons to reduce model complexity and speed up calculations, demonstrates that removing connections with small weights maintains performance [29]. Exp. 1 shows a similar trend.

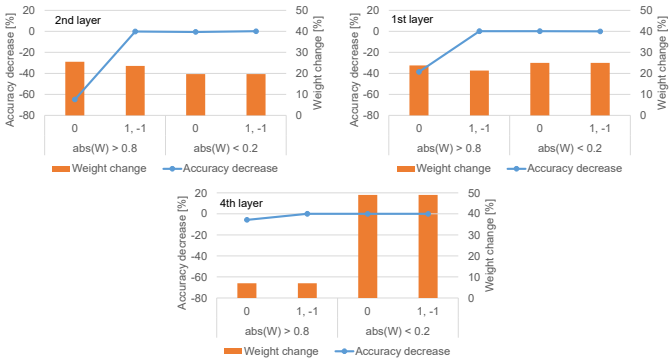


Fig. 3 Weight change vs. accuracy change (NN)

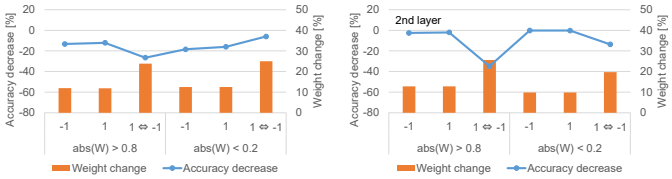


Fig. 4 Weight change vs. accuracy change (BN)

Furthermore, compared to the first and second layers, the fourth layer has a smaller impact on accuracy when the larger weights are changed to 0. This is thought to be because the proportion of large weights in the fourth layer is smaller than in the first and second layers (Fig. 5). Furthermore, from Fig. 4, when the sign of the weights in the first and second layers of the BN are inverted, there is the most significant impact on accuracy. This is considered to be because the proportion of the number of changed weights is large, in addition to the influence of the weight magnitude mentioned above.

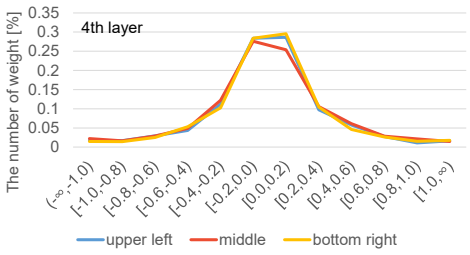


Fig. 5 Weight distribution

3.2.2 Experiment 2

(1) Faults of Row Direction in the First Layer

Changes to the weights in the row direction (total 784 rows) of the first layer resulted in a decrease in accuracy as the number of rows modified increased. Fig. 6 shows the rate of accuracy change when the weight values of 100 rows were changed.

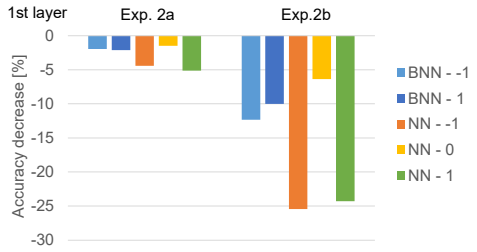


Fig. 6 Accuracy change (1st layer)

It was observed that the impact on accuracy was more significant in Exp.2b than in Exp.2a. When the fault locations were dispersed (Exp. 2a), the locations of the noisy pixels caused by the faults were also distributed, and the overall image did not change significantly. On the other hand, when the fault locations were concentrated (Exp. 2b), the noisy pixels were concentrated in a specific location, potentially causing a greater impact on recognition accuracy. Furthermore, for the standard NN, the impact on accuracy was more pronounced when faulty weights were +1 or -1 compared to 0. From the results of Exp. 1, it is considered that the impact on accuracy will be greater for larger weights, so this result is reasonable. Additionally, when comparing the impact of faulty weights being +1 or −1, the accuracy of the standard NN was more affected by weight changes than that of the BN. Since the original weights of the BNN are only +1 and −1, there are only two possible weight change patterns: from +1 to −1, or from −1 to +1. In contrast, in the standard NN, real values in the range of −1.0 to +1.0 are used for the original weights, so various patterns of weight change are possible. It is considered that the difference in the weight change pattern is the cause of the difference in the impact on accuracy.

Table 1 Accuracy variations

	BNN							
	Exp. 2a				Exp. 2b			
# of rows	Min	Max	diff.	ave.	Min	Max	diff.	ave.
1	96.98	97.12	0.14	97.05	96.98	97.12	0.14	97.05
5	97.02	97.14	0.13	97.07	96.91	97.11	0.20	97.01
10	96.97	97.14	0.17	97.03	96.89	97.12	0.23	97.00
20	96.88	97.11	0.23	97.00	96.91	97.13	0.22	97.01
30	96.58	97.03	0.45	96.85	95.68	97.06	1.38	96.47
50	96.34	96.92	0.58	96.70	94.86	96.70	1.84	95.72
100	94.71	95.53	0.82	95.07	81.72	89.96	8.25	86.22
	NN							
	Exp. 2a				Exp. 2b			
# of rows	Min	Max	diff.	ave.	Min	Max	diff.	ave.
1	97.48	97.54	0.06	97.51	97.48	97.54	0.06	97.51
5	97.44	97.56	0.12	97.51	97.37	97.55	0.19	97.49
10	97.40	97.58	0.18	97.50	97.34	97.59	0.24	97.48
20	97.24	97.60	0.36	97.40	97.33	97.52	0.19	97.42
30	97.12	97.48	0.36	97.28	95.90	97.52	1.62	96.83
50	96.80	97.33	0.53	97.00	93.72	97.23	3.51	95.24
100	92.78	95.21	2.42	93.93	69.78	84.71	14.93	79.29

Table 1 shows the minimum and maximum accuracy from five experiments. From Table 1, as the number of rows changed increases, the variability in recognition accuracy increases. Therefore, it is thought that the impact on accuracy differs depending on the location of the weight change, i.e., the pixel position affected by the fault. This is one of the issues to be addressed in the future.

(2) Faults of Column Direction in the Fourth Layer

Fig. 7 shows the rate of accuracy change depending on the number of columns modified when the column direction weights in the fourth layer were changed to 1 using the method of Exp. 2a. For each additional column whose weight was changed, accuracy dropped by approximately 10%. When all 10 columns had weight faults, accuracy dropped by approximately 90%. Each column corresponds to a digit from 0 to 9, and it is thought that accuracy drops by approximately 10% because columns with changed weights fail to recognize the corresponding digits correctly. Similar results were obtained with the method of Exp.2b and other weight faults.

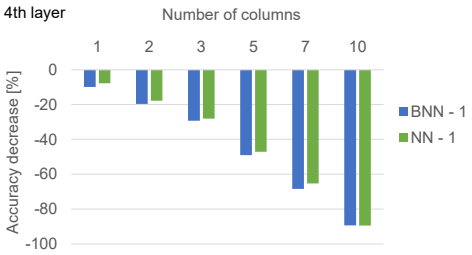


Fig. 7 Accuracy change (4th layer)

(3) Faults of Other Row and Column Directions

Fig. 8 shows the rate of change in accuracy when changing the weight values of 800 rows/columns out of 6144 rows/columns for the 1st, 2nd, and 4th layers. As with (1), changing the weights has a greater impact on accuracy for the standard NN than for the BN. However, the impact on accuracy is smaller than in (1), where the weights of 784 rows were changed. The ratio of the number of rows (columns) changed to the total number of rows (columns) is 13.0%, while that of (1) is 12.8%, which is a difference of only about 0.2%. However, the impact on accuracy is clearly greater in (1).

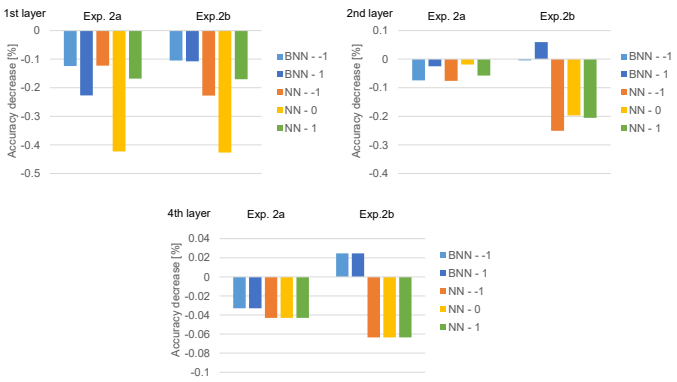


Fig. 8 Accuracy change (row and column directions of other layers)

This is thought to be due to the relationship between the images used in MNIST and the array that stores the weights. Fig. 9 shows the relationship between the array size that stores the pixels and the weights. The number of pixels in the images used in MNIST is 28 × 28 = 784, which matches the number of rows in the weight array in the first layer, so the 784 rows of data are thought to be weights corresponding to each pixel in the image. In addition, in the BN, there are 2048 nodes (neurons) for each pixel, and if it is assumed that each node has three outputs, the total weight becomes 2048 × 3 = 6144. Therefore, since it matches the number of columns in the weight array in the first layer, each column of the 6144 columns of weights is thought to correspond to each output of one pixel. In other words, changing all the weights in one row out of the 784 rows changes all the weights corresponding to one pixel, so the impact of the fault is concentrated on one pixel. Therefore, since the impact of the fault on the image is large, the impact on the recognition accuracy is also large. On the other hand, changing all the weights in one column out of the 6144 columns changes only one of the 6144 weights for each pixel. Therefore, since it only has a very small effect on the entire image, it is thought that the impact on accuracy is small.

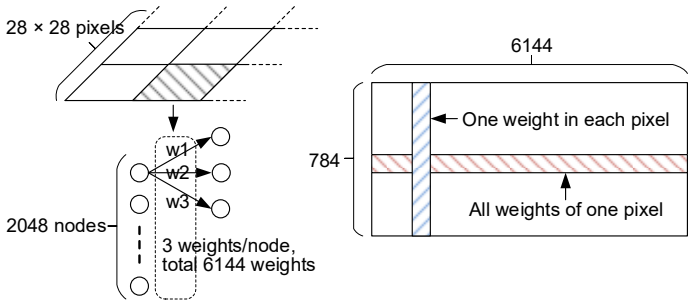


Fig. 9 Image pixels and array data

4. Comparison of NN and BNN

This study assumed stuck-at faults in memory cells and their surrounding circuits, and investigated the impact on recognition accuracy when weights were changed to +1, −1, or 0. Under the experimental conditions in Section 3, the weights of the selected rows or columns of the array were changed. In this case, at most 50% of the weights changed in the BNN (e.g., +1 → −1), whereas in the NN nearly 100% of the weights changed (e.g., +1.0 to −0.9 → −1). Therefore, the number of faults inserted (number of weight changes) is different. Therefore, we compared the impact of faults on the accuracy of the NN and BNN in terms of the rate of decrease in recognition accuracy relative to the number of weight changes ((accuracy decrease rate)/(weight change rate)). The results for the weight changes of 100 rows in Fig. 6 are as follows: NN > BNN (Table 2).

Therefore, the rate of decrease in recognition accuracy relative to the weight change rate is smaller for the BNN than for the NN.

Table 2 Ratio of accuracy decrease and weight change

	Expt. 2a		Expt. 2a	
	NN	BNN	NN	BNN
Accuracy decrease rate [%]	5	2	25	10
Weight change rate [%]	100	50	100	50
Accuracy decrease rate / Weight change rate	0.05	0.04	0.25	0.2

In this study, we assume the occurrence of stuck-at faults, so the weight after the fault will be either 0 or 1. In addition, weight -1 in the BNN is coded as 0 ((−1, +1) is coded as (0, 1)). Therefore, even if a stuck-at-1 (0) fault occurs in an area that is originally a weight of +1 (−1), the fault is masked and the weight is not affected (recognition accuracy does not decrease). On the other hand, in the standard NN, for example, if a stuck-at-1 (0) fault occurs in an area that is originally a large weight of +0.8 to +1.0 (−0.8 to −1.0), the weight value will change, unlike in the BNN, and recognition accuracy will be affected. In particular, as shown in Exp.1, the impact on accuracy is large when the value of a large weight becomes smaller, so for example, if a stuck-at-0 fault occurs in a large weight of +0.8 to +1.0 (−0.8 to −1.0), recognition accuracy may decrease significantly. Therefore, it is considered that the BNN is advantageous in the case of faults in which the value after the weight change matches the original value.

Furthermore, the BNN has only two fault types: 1 → 0 (−1) and 0 (−1) → 1, whereas the NN has a greater number of fault types, as real values in the range of −1.0 to +1.0 change to 0 or 1. As a result, the NN is likely to have a greater number of faulty weights (weights whose values change) than the BNN.

From the above, it is considered that hardware implementation using the BNN is more advantageous than using the NN, even when hardware faults are assumed.

5. Conclusions

This study compared the impact of weight changes due to stuck-at faults on recognition accuracy in the BNN and the standard NN. As a result, for example, when the weight values of 100 consecutive rows in the first layer were set to +1 and −1 by the fault, the accuracy of the standard NN decreased by about 24.9%, while that of the BNN decreased by only about 11.2%. It was also confirmed that the impact of weight changes on accuracy in the BNN was smaller than that of the standard NN for other weight changes. In addition, changes in the weights of the hidden layer have less impact on recognition accuracy. This is due to the relationship between the pixels of the image and the array that stores the weights of the neurons. The decrease in recognition accuracy relative to the weight change ratio was also smaller in the BNN than in the standard NN. Furthermore, since weights in the BNN are coded as 1 and 0, the effect of stuck-at faults may be masked, so it is considered that the decrease in recognition accuracy due to faults is smaller than in the standard NN. We also confirmed that in the standard NN, large weight changes (faults) have a greater impact on accuracy than small weights.

Future work includes the relationship between the number of rows/columns (or the number of faults) of the changed weights and the accuracy, the dependency of accuracy on the location where the weights are changed (edge or center of the image), and experiments on other datasets.

References

[1] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, issue 11, pp.2278 – 2324, November 1998.

[2] S. Ren, K. He, R. Girshick and J. Sun. Faster, “Faster R-CNN: Towards real-time object detection with region proposal networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, issue 6, pp.1137 – 1149, June 2017.

[3] L. Deng, G. Hinton and B. Kingsbury, “New types of deep neural network learning for speech recognition and related applications: an overview,” 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, May 2013.

[4] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D, Yu, “ Convolutional neural networks for speech recognition, ” IEEE/ACM Transactions on audio, Speech, and Language Processing, vol. 22, no. 10, October 2014.

[5] T. Yang, Y. Chen, J. Emer and V. Sze, “A method to estimate the energy consumption of deep neural networks,” 2017 51st Asilomar Conference on Signals, Systems, and Computers. October 2017.

[6] N.M. Botros and M. Abdul-Aziz, “Hardware implementation of an artificial neural network,” IEEE International Conference on Neural Networks, pp.1252-1257 March 1993.

[7] S. Jung and S. Kim, “Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems,” IEEE Transactions on Industrial Electronics, vol. 54, no. 1, pp.265-271, February 2007

[8] J. Misra and I. Saha, “Artificial neural networks in hardware: A survey of two decades of progress,” Neurocomputing, vol. 74, issues 1–3, pp. 239-255, December 2010.

[9] E. Z. Mohammed and H. K. Ali, “Hardware implementation of artificial neural network using field programmable gate array,” International Journal of Computer Theory and Engineering, vol. 5, no. 5, pp. 780-783, October 2013

[10] M. Courbariaux, Y. Bengio and J. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” Neural Information Processing Systems 28, 2015.

[11] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio: “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1,” Computing Research Repository, pp. 1-11, March 2016.

[12] H. Qin., R. Gong, X. Liu, X. Bai., J. Song and, N. Sebe, “Binary neural networks: A survey,” Pattern Recognition vol. 105 pp. 1-14, September 2020.

[13] K. Hwang and W. Sung “Fixed-point feedforward deep neural network design using weights +1, 0, and -1,” IEEE Workshop on Signal Processing Systems, October 2014.

[14] J. Nijhuis, B. Hofflinger, A. van Schaik and L. Spaanenburg, “Limits to the fault-tolerance of a feedforward neural network with learning,” 20th International Symposium on Fault-Tolerant Computing, pp. 228-235, June 1990

[15] C.H. Sequin and R.D. Clay, “Fault tolerance in artificial neural networks,” International Joint Conference on Neural Networks, pp. I703-I708, June 1990.

[16] Z.H. Zhou, S.F. Chen and Z.Q. Chen, “Evolving fault-tolerant neural networks,” Neural Computing and Applications, vol. 11, pp. 156–160, June 2003.

[17] C. Torres-Huitzil and Girau, “Fault and error tolerance in neural networks: A review,” IEEE Access, vol. 5, pp. 17322-17341, August 2017

[18] 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,” Proceedings of 55th ACM/ESDA/IEEE Design Automation Conference, June 2018.

[19] E. Ozen and A. Orailoglu, “Shaping resilient AI hardware through DNN computational feature exploitation,” IEEE Design and Test, vol. 40, no. 2, pp. 59-66, March/April 2023.

[20] C. C. Aggarwal, “Neural networks and deep learning: A textbook,” Second Ed., Springer, 2023,

[21] H. Zhang, D. Chen and S. Ko, “New flexible multiple-precision multiply-accumulate unit for deep neural network training and inference,” IEEE Transactions on Computers, vol. 69, issue: 1, pp. 26-38, January 2020.

[22] M.B. Tahoori, S. Mitra, S. Toutounchi and E.J. McCluskey, “Fault grading FPGA interconnect test configurations,” Proceedings of International Test Conference, October 2002.

[23] M. Rebaudengo, M.S. Reorda and M. Violante, “A new functional fault model for FPGA application-oriented testing,” Proceedings of 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, November 2002.

[24] G. Asadi and M.B. Tahoori, “Soft error rate estimation and mitigation for SRAM-based FPGAs,” F Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays, pp. 149-160, February 2005

[25] A. Lesea, S. Drimer, J.J. Fabula, C. Carmichael and P. Alfke, “The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs,” IEEE Transactions on Device and Materials Reliability, vol. 5, issue 3, pp. 317-328, September 2005.

[26] C.L. Hsu and T.H. Chen, “Built-in self-test design for fault detection and fault diagnosis in SRAM-based FPGA,” IEEE Transactions on Instrumentation and Measurement, vol. 58, issue 7, pp. 2300 – 2315, July 2009.

[27] https://github.com/itayhubara/BinaryNet.pytorch

[28] 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, 1998.

[29] M. Zhu and S. Gupta, “To prune, or not to prune: exploring the efficacy of pruning for model compression”, ICLR Workshop, February 2018.