Article Information

Title	Approximated Triple Modular Redundancy of Convolutional Neural Networks Based on Residual Quantization
Authors	Yamato Saikawa, Yoichi Tomioka
Citation	2024 IEEE 17th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Kuala Lumpur, Malaysia, 2024, pp. 302-309, doi: 10.1109/MCSoC64144.2024.00057.
Copyright	
Note	This is the author's accepted version of the paper published in *Proceedings of IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (2024)*. The final version is available at IEEE Xplore DOI: https://doi.org/10.1109/MCSoC64144.2024.00057

Approximated Triple Modular Redundancy of Convolutional Neural Networks Based on Residual Quantization

1st Yamato Saikawa and 2nd Yoichi Tomioka
Graduate School of Computer Science and Engineering
The University of Aizu
Aizu-Wakamatsu City, Fukushima, Japan
{d8261107,ytomioka}@u-aizu.ac.jp

***Abstract*—Triple modular redundancy (TMR) is a fault-tolerant system that detects faults and allows processing to continue by employing three identical modules. However, TMR introduces challenges such as tripled circuit area and increased power consumption. In this paper, we propose an approximated TMR (ATMR), which consists of the original module and two smaller modules achieved through quantization, specifically tailored for convolutional neural networks (CNNs). For evaluation, we conduct experiments using ResNet-20 to simulate fault recovery through linear and residual quantization when the original module fails. The results demonstrate that ATMR maintains high inference accuracy across many layers, even when employing low-bit linear or residual quantization for recovery.**

***Index Terms*—convolutional neural network, quantization, triple modular redundancy, fault detection, restoration, fault tolerance**

I. Introduction

Edge AI systems based on deep neural networks (DNNs) have started being used in various fields and affect our lives closely. The DNNs that are most utilized as current edge AIs are based on Artificial neural networks (ANNs). It mainly calculates matrix multiplication with real or integer values and achieves high inference accuracy for data analysis, classification, pattern recognition, etc.

Convolutional neural networks (CNNs) [1] can classify multidimensional data more accurately than general ANNs. It is reported that the CNNs of deeper layers have more problems with gradient loss or explosion. Therefore, finding the appropriate parameters for such CNNs is difficult through back-propagation-based training even though they have the potential to improve accuracy. To solve the problems, Residual Network (ResNet) [2] is conceived. This method features a skip connection that has connected identity mapping and residual mapping, and it is called residual learning. Thanks to it, CNNs are able to consist of many layers and obtain very high inference accuracy. However, CNN inference is computationally expensive and requires a dedicated circuit for real-time processing. Such a circuit requires high power consumption and a larger circuit area to achieve higher performance since to achieve faster CNN inference, it is necessary to achieve highly parallel computation. In particular, dot products, including convolution operations, request many resources such as logic gates, registers, etc. Therefore, CNN accelerators of low power and small size are needed in order for CNNs to be utilized on edge devices that have serious restrictions such as power consumption and heat dissipation.

To solve the size and power consumption problems of CNNs, there have been works of model compression and several of them have shown high compression ratios and inference accuracy. For example, the method called quantization [3]–[5] can reduce the original bit width to lower. Keeping the structure of the original model, the size and computing resources are able to be compressed owing to that.

On the other hand, there is report [6] that circuits may have crises, such as unintentional behavior due to aging or physical failure. Especially, malfunction of automatic driving systems, robots, medical devices, and so on have possible serious accidents affect human life. Thus, AI in safety-critical edge systems should be realized to detect malfunctions and prevent accidents in advance.

As the existing method to keep high reliability, there is triple modular redundancy (TMR) that consists of three modules of the same circuit. It can detect the fault and continue working normally when one of the three is out of order as it gathers outputs from three circuits and selects their majority. Although TMR can get robustness of systems, it is difficult to use large-scale systems in edge devices like chips with small sizes because implementation costs such as power consumption and circuit area increase by a factor of three or more. In the current situation, the TMR system in a CNN is hard to adapt to the implementation of small-size chips. Thus, we aim to relax the problems and achieve fault tolerance with a smaller area and lower power.

From the above, we propose an approximated TMR (ATMR) based on quantization for CNNs. In this method, two redundant modules are replaced in quantized them, and they play a role in fault detection and substitute of inference if the original module is malfunctioning. Moreover, we also propose restoration by two methods that are based on linear

This work was supported by JST, PRESTO Grant Number JPMJPR22C7, Japan.

quantization and residual quantization. Hence, it is considered that the problems that the conventional TMR has can be relaxed while the proposed method keeps high reliability.

Our contributions to this thesis are summarized as follows:

- We propose ATMR(8,m,m) that consists of the original convolutional layer and two m-bit quantized convolutional layers. This work extends approximated dual modular redundancy (ADMR) [7], which is based on linear quantization. While ADMR cannot restore outputs upon failure, the proposed ATMR enables continuous inference even when faults are detected. By partially applying residual quantization, ATMR achieves an accuracy comparable to a non-faulty model, despite requiring less computational cost than the conventional TMR approach.
- We perform output restoration experiments of ATMR(8,m,m) which consists of 8-bit and m-bit convolutional layers ($m = \{3, 4, 5, 6, 7\}$) for 8-bit ResNet-20 trained with CIFAR-10.
- In software simulation, the output restoration of ResNet-20 based on linear quantization achieves that ATMR(8,3,3) and ATMR(8,4,4) recover the inference accuracy to near 91.8% which the original has when their area under the curve (AUC) of fault detection is 1 in many layers. Moreover, we demonstrate that residual quantization helps improve the accuracy in layers where the accuracy drops by linear quantization-based ATMR.

II. Related Works

Reference [8] proposes a fault-tolerance method based on N-version programming. In this approach, multiple neural networks are independently trained with different parameters, architectures, or subsets of the training data and executed in parallel. The study evaluated the method using the MNIST and CIFAR-10 datasets, showing that parallel networks achieved higher accuracy than individual networks. However, the method faces challenges, such as performance degradation caused by permanent circuit faults. Reference [9] presents a technique for generating quantized neural networks with low-bit precision and performing multiply-accumulate operations with a guard bit to enhance fault tolerance. Similarly, Reference [10] proposes constructing dual modular redundancy (DMR) at the neuron level to mitigate performance degradation under single-bias attacks. However, these methods have limitations: they cannot address faults affecting entire layers because they only duplicate individual neurons, nor can they handle faults other than increased delays in multiply-accumulate operations. Reference [11] introduces an approximated DMR unit that employs ternary convolutional layers and random forests to significantly reduce circuit area and computational costs while detecting a variety of faults. However, this method is not directly applicable to general CNN models with wide bit widths, limiting its versatility in broader applications.

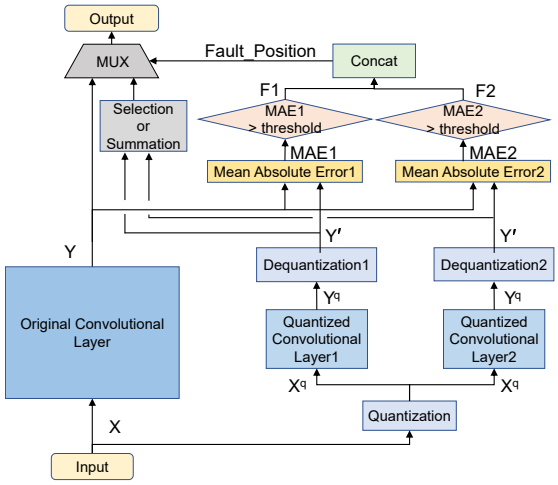

Fig. 1. Proposed ATMR.

III. Proposed ATMR Method

A. ATMR Outline

The outline of the proposed ATMR is shown in Figure 1. $\boldsymbol{X}$ is a input tensor that consists of $C_{in} \times H \times W$, where C_{in} is the number of input channel, H is the height of each channel and W is the width.

First of all, an input $\boldsymbol{X}$ enters the Original Convolutional Layer and the Quantization module. The Original Convolutional Layer is the main layer to infer and perform convolution operations with original-bit input and weight. After the calculation, it outputs $\boldsymbol{Y}$ that consists of $C_{out} \times H \times W$ elements. It is denoted by

$$\boldsymbol{Y} = \mathrm{Conv}\left(\boldsymbol{X}, \boldsymbol{W}\right). \tag{1}$$

Defining c, h, and w as the indices of tensor elements, respectively, the output of $\boldsymbol{Y}_{c,w,h}$ is calculated as follows:

$$\boldsymbol{Y}_{c,h,w} = \sum_{i=0}^{C_{in}-1} \sum_{j=0}^{K-1} \sum_{k=0}^{K-1} \boldsymbol{X}_{i,h+j-\lfloor \frac{K}{2} \rfloor, w+k-\lfloor \frac{K}{2} \rfloor} \cdot \boldsymbol{W}_{c,i,j,k}, \tag{2}$$

where K is the size of the kernel which the layer has as weights, $\boldsymbol{W}$ is a tensor as a weight that consists of $C_{out} \times C_{in} \times K \times K$ elements.$\boldsymbol{W}_c$ represents c-th weight filter in the weight tensor $\boldsymbol{W}$.

The quantization module is a module to quantize $\boldsymbol{X}$ with the original bits to $\boldsymbol{X^q}$ with the lower bits. The module has α as the scaling factor and γ as the zero-point to compress to a specified bit width with a minimum value of 0. Moreover, the quantization is performed for each channel. Thus, $\boldsymbol{X^q}$, α and γ are calculated as follows:

$$\boldsymbol{X^q} = \mathrm{round}\left(\alpha(\boldsymbol{X} - \gamma)\right), \tag{3}$$

$$\alpha = \frac{2^m - 1}{\max\{\boldsymbol{X}\} - \min\{\boldsymbol{X}\}} \tag{4}$$

,

$$\gamma = \min\{\boldsymbol{X}\}, \tag{5}$$

where m is the bits number after the quantization, $\max$ is the function to return the maximum value from $\boldsymbol{X}$ and $\min$ is the function to return the minimum value.

After the quantization, $\boldsymbol{X^q}$ enters two Quantized Convolutional Layers. They are modules to calculate convolution with lower-bit quantized weights and play a role in fault detection and restoration when the inference module is broken. Additionally, they have the same $\boldsymbol{W^q}$ that is generated by linear quantization based on $\boldsymbol{W}$. $\boldsymbol{W^q}$ and scaling factor β for it are calculated as follows:

$$\boldsymbol{W^q} = \mathrm{round}\left(\beta \boldsymbol{W}\right) \tag{6}$$

$$\beta = \frac{2^{m-1}-1}{\max\{|\boldsymbol{W}|\}} \tag{7}$$

Let $\boldsymbol{Y^q}$ be the result of the convolutional operation of the two quantization tensors, $\boldsymbol{X^q}$ and $\boldsymbol{W^q}$. It is calculated by

$$\boldsymbol{Y^q} = Conv(\boldsymbol{X^q}, \boldsymbol{W^q}). \tag{8}$$

$\boldsymbol{Y^q}$ has different bits from the original layer. The value used for fault detection and output restoration is required to be the same bit width as $\boldsymbol{Y}$. Therefore, the approximated value, $\boldsymbol{Y'}$ is generated by dequantizing in each Dequantization module. The dequantization of $\boldsymbol{X^q}$ and $\boldsymbol{W^q}$ can be calculated as follows:

$$\boldsymbol{X} \approx \frac{\boldsymbol{X^q}}{\alpha} + \gamma \tag{9}$$

$$\boldsymbol{W} \approx \frac{\boldsymbol{W^q}}{\beta} \tag{10}$$

Therefore, the convolutional operation of their approximated tensors also can be calculated as follows:

$$\begin{aligned}
&\boldsymbol{Y_{c,h,w}} \approx \\
&\sum_{i=0}^{C_{in}-1}\sum_{j=0}^{K-1}\sum_{k=0}^{K-1}\left(\frac{\boldsymbol{X}^q_{i,h+j-\lfloor\frac{K}{2}\rfloor,w+k-\lfloor\frac{K}{2}\rfloor}}{\alpha}+\gamma\right)\cdot\frac{\boldsymbol{W}^q_{c,i,j,k}}{\beta} \\
&= \frac{1}{\alpha\beta}\sum_{i=0}^{C_{in}-1}\sum_{j=0}^{K-1}\sum_{k=0}^{K-1}\boldsymbol{X^q}_{i,h+j-\lfloor\frac{K}{2}\rfloor,w+k-\lfloor\frac{K}{2}\rfloor}\cdot\boldsymbol{W^q}_{c,i,j,k} + \\
&\qquad \gamma\sum_{i=0}^{C_{in}-1}\sum_{j=0}^{K-1}\sum_{k=0}^{K-1}\frac{\boldsymbol{W^q}^q_{c,i,j,k}}{\beta} \\
&= \frac{1}{\alpha\beta}\boldsymbol{Y^q}_c + \gamma\cdot\mathrm{sum}\left(\boldsymbol{W}_c\right)
\end{aligned} \tag{11}$$

From the above, let α' be $\alpha\beta$ as the new scaling factor and γ' be $\gamma\cdot\mathrm{sum}\left(\boldsymbol{W_c}\right)$ as the new zero point, $\boldsymbol{Y'}$ as the dequantized tensor is calculated as follows:

$$\boldsymbol{Y'} = \frac{\boldsymbol{Y^q}}{\alpha'} + \gamma' \tag{12}$$

Secondly, the fault occurrence of each convolutional layer is predicted, and output restoration is performed. Based on the fault detection method in [7], mean absolute errors (MAEs) between Y and two Y' are calculated as follows:

$$MAE = \frac{\|\boldsymbol{Y}-\boldsymbol{Y'}\|_1}{C\cdot H\cdot W} \tag{13}$$

After that, the fault is predicted by checking whether the values exceed a threshold. The results of Mean Absolute Error 1 and 2 are denoted by $MAE1$ and $MAE2$, respectively. Each value is fed into a corresponding module that outputs 1 if the input exceeds the threshold or 0 otherwise. This process is formulated as a two-class classification problem, where each layer is classified as either being in a fault state (positive class) or a normal state (negative class) based on its MAE. The MAE between the output Y from the faulty layer and Y' from the normal layer is treated as a positive sample, regardless of any inferred class changes due to the fault in that layer. Conversely, the MAE from the normal layer's output is considered a negative sample.

The number of correctly classified positive samples is defined as True Positives (TP), while the number of correctly classified negative samples is defined as True Negatives (TN). Similarly, the number of misclassified positive samples is defined as False Negatives (FN), and the number of misclassified negative samples is defined as False Positives (FP). The True Positive Rate (TPR) and False Positive Rate (FPR) are calculated as follows:

$$TPR = \frac{TP}{FN+TP} \tag{14}$$

$$FPR = \frac{FP}{TN+FP} \tag{15}$$

Additionally, a simulation using several image data is conducted, and based on that, thresholds that satisfy the specified FPR are obtained from the MAE distribution in normal and fault layers.

After the conditional branch, two binary results are combined as $Fault\_Position$ with 2 bits to identify fault layers, and it is used as the control signal of the multiplexer. For example, $Fault\_Position$ becomes $\{1,1\}$ when the original Convolutional Layer has a fault, and others are normal. When all layers work normally, $\{0,0\}$ is output. When one of the Quantized Convolutional Layers is out of order, $\{1,0\}$ or $\{0,1\}$ are output. Finally, the multiplexer determines which output of layers to send to next based on the control signal. If the approximated output is determined, a selection of one side or an average of two is used. Furthermore, there are two types of approximated output. The first is to use the tensor calculated by Eq. 12. The second is to use the tensor generated by residual quantization and dequantization.

B. Residual Quantization for Output Restoration

Residual quantization [12] iteratively quantizes the quantization error, and the values are used for narrowing the error range during dequantization to the original value. Thus, the method can approximate values more closely to the original compared to linear quantization in exchange for calculation costs of iterative quantization.

Assuming k iterations and input $\boldsymbol{X}$, the quantized $\boldsymbol{X^q}_{(k)}$, $\alpha_{(k)}$ as the scaling factor and $\gamma_{(k)}$ as the zero point are calculated as follows:

$$\boldsymbol{X^q}_{(k)} = \text{round}\left(\alpha_{(k)}(\Delta\boldsymbol{X^q}_{(k-1)} - \gamma_{(k)})\right) \tag{16}$$

$$\alpha_{(k)} = \frac{2^m - 1}{\max\{\Delta\boldsymbol{X^q}_{(k-1)}\} - \min\{\Delta\boldsymbol{X^q}_{(k-1)}\}} \tag{17}$$

$$\gamma_{(k)} = \min_{\Delta\boldsymbol{X^q}_{(k-1)}}, \tag{18}$$

where $\Delta\boldsymbol{X^q}_{(k-1)}$ means a quantization error between $\Delta\boldsymbol{X^q}_{(k-2)}$ and $\boldsymbol{X^q}_{(k-1)}$, and it is calculated as follows:

$$\Delta\boldsymbol{X^q}_{(k)} = \begin{cases} \boldsymbol{X} & \text{if } k = 0, \\ \Delta\boldsymbol{X^q}_{(k-1)} - \frac{\boldsymbol{X^q}_{(k)}}{\alpha_{(k)}} - \gamma_{(k)} & \text{if } k \geq 1 \end{cases} \tag{19}$$

Algorithm 1 Calculation of $\boldsymbol{X^q}$ with N iterations.

Input: $\boldsymbol{X}$
Output: $\boldsymbol{X^q}$
  for $r = 0$ to N **do**
    if $r = 0$ **then**
      $\Delta\boldsymbol{X^q}[r] \leftarrow \boldsymbol{X}$
    else
      $\gamma[r] \leftarrow \min(\Delta\boldsymbol{X^q}[r-1])$
      $\alpha[r] \leftarrow 2^m - 1/\{\max(\Delta\boldsymbol{X^q}[r-1]) - \min(\Delta\boldsymbol{X^q}[r-1])\}$
      $\boldsymbol{X^q}[r] \leftarrow \text{round}\left(\alpha[r] * \Delta\boldsymbol{X^q}[r-1] - \gamma[r]\right)$
      $\Delta\boldsymbol{X^q}[r] \leftarrow \Delta\boldsymbol{X^q}[r-1] - \boldsymbol{X^q}[r]/\alpha[r] - \gamma[r]$
    end if
  end for
  return $\boldsymbol{X^q}$

Collectively, the above equations from 16 to 19 are executed and the items are obtained such as Algorithm 1. Therefore, assuming that iterations is N, the result of input residual quantization becomes $\boldsymbol{X^q} = \{\boldsymbol{X^q}_{(1)}, \boldsymbol{X^q}_{(2)}, \boldsymbol{X^q}_{(3)}, \ldots, \boldsymbol{X^q}_{(N-1)}, \boldsymbol{X^q}_{(N)}\}$, and it is equivalent to linear quantization if $N = 1$.

Furthermore, assuming k iterations and weight $\boldsymbol{W}$, the quantiazed $\boldsymbol{W^q}_{(k)}$ and $\beta_{(k)}$ as the scaling factor are calculated as follows:

$$\boldsymbol{W^q}_{(k)} = \text{round}\left(\beta_{(k)}\Delta\boldsymbol{W^q}_{(k-1)}\right) \tag{20}$$

$$\beta_{(k)} = \frac{2^{m-1} - 1}{\max\{|\Delta\boldsymbol{W^q}_{(k-1)}|\}}, \tag{21}$$

where $\Delta\boldsymbol{W^q}_{(k-1)}$ means a quantization error between $\Delta\boldsymbol{W^q}_{(k-2)}$ and $\boldsymbol{W^q}_{(k-1)}$, and it is calculated as follows:

$$\Delta\boldsymbol{W^q}_{(k)} = \begin{cases} \boldsymbol{W} & \text{if } k = 0, \\ \Delta\boldsymbol{W^q}_{(k-1)} - \frac{\boldsymbol{W^q}_{(k)}}{\beta_{(k)}} & \text{if } k \geq 1 \end{cases} \tag{22}$$

Collectively, the above equations from 20 to 22 are executed and the items are obtained such as Algorithm 2. Therefore, assuming that iterations is N', the result of residual quantization for weight becomes $\boldsymbol{W^q} = \{\boldsymbol{W^q}_{(1)}, \boldsymbol{W^q}_{(2)}, \boldsymbol{W^q}_{(3)}, \ldots, \boldsymbol{W^q}_{(N'-1)}, \boldsymbol{W^q}_{(N')}\}$. Let $\boldsymbol{Y^q}$ be the result of the convolutional operation of the two quantization tensors, $\boldsymbol{X^q}_{(r)}$ and $\boldsymbol{W^q}_{(r')}$, the calculation of $\boldsymbol{Y^q}_{(r,r')}$ is the following:

Algorithm 2 Calculation of $\boldsymbol{W^q}$ with N' iterations.

Input: $\boldsymbol{W}$
Output: $\boldsymbol{W^q}$
  for $r' = 0$ to N' **do**
    if $r = 0$ **then**
      $\Delta\boldsymbol{W^q}[r'] \leftarrow \boldsymbol{W}$
    else
      $\beta[r'] \leftarrow 2^{m-1} - 1/\max(\text{abs}(\Delta\boldsymbol{W^q}[r'-1]))$
      $\boldsymbol{W^q}[r'] \leftarrow \text{round}\left(\beta[r'] * \Delta\boldsymbol{W^q}[r'-1]\right)$
      $\Delta\boldsymbol{W^q}[r'] \leftarrow \Delta\boldsymbol{W^q}[r'-1] - \boldsymbol{W^q}[r']/\beta[r']$
    end if
  end for
  return $\boldsymbol{W^q}$

$$\boldsymbol{Y^q}_{(r,r')} = \text{Conv}\left(\boldsymbol{X^q}_{(r)}, \boldsymbol{W^q}_{(r')}\right) \tag{23}$$

This operation is performed for all combination of quantized tensors that are in $\boldsymbol{X^q}$ and $\boldsymbol{W^q}$ such as Algorithm 3. Thus, $\boldsymbol{Y^q}$ becomes a tensor that consists of $N \times N' \times C_{out} \times H \times W$.

Algorithm 3 Calculation of $\boldsymbol{Y^q}$ from N iterations $\boldsymbol{X^q}$ and N' iterations $\boldsymbol{W^q}$

Input: $\boldsymbol{X^q}, \boldsymbol{W^q}$
Output: $\boldsymbol{Y^q}$
  for $r = 1$ to N **do**
    for $r' = 1$ to N' **do**
      $\boldsymbol{Y^q}[r][r'] \leftarrow \text{Conv}(\boldsymbol{X^q}[r], \boldsymbol{W^q}[r'])$
    end for
  end for
  return $\boldsymbol{Y^q}$

As same as linear quantization, $\boldsymbol{Y^q}_{(r)}$ has different bit width from $\boldsymbol{Y}$. Therefore, dequantization is performed for the output restoration. The dequantization of $\boldsymbol{X^q}_{(r)}$ and $\boldsymbol{W^q}_{(r)}$ calculated by residual quantization can be calculated as follows:

$$\boldsymbol{X} \approx \sum_{r=1}^{N} \frac{\boldsymbol{X}_{(r)}}{\alpha_{(r)}} + \sum_{r=1}^{N} \gamma_{(r)} + \Delta\boldsymbol{X}_{(N)} \tag{24}$$

$$\boldsymbol{W} \approx \sum_{r'=1}^{N'} \frac{\boldsymbol{W}_{(r')}}{\beta_{(r')}} + \Delta\boldsymbol{W}_{(N')} \tag{25}$$

Based on Eq. 24 and 25, the approximated value of $\boldsymbol{Y}$ also can be calculated as follows:

$$
\begin{aligned}
&\boldsymbol{Y_{c,h,w}} \approx \\
&\sum_{r=1}^{N}\sum_{r'=1}^{N'}\sum_{i=0}^{C_{in}-1}\sum_{j=0}^{K-1}\sum_{k=0}^{K-1} \\
&\left(\frac{\boldsymbol{X}^{q}_{(r),i,h+j-\lfloor\frac{K}{2}\rfloor,w+k-\lfloor\frac{K}{2}\rfloor}}{\alpha_{(r)}}+\gamma_{(r)}\right)\cdot\frac{\boldsymbol{W^q}_{(r'),c,i,j,k}}{\beta_{(r')}} \\
&=\sum_{r=1}^{N}\sum_{r'=1}^{N'}\frac{1}{\alpha_{(r)}\beta_{(r')}}\sum_{i=0}^{C_{in}-1}\sum_{j=0}^{K-1}\sum_{k=0}^{K-1} \\
&\qquad\boldsymbol{X^q}_{(r),i,h+j-\lfloor\frac{K}{2}\rfloor,w+k-\lfloor\frac{K}{2}\rfloor}\cdot\boldsymbol{W^q}_{(r'),c,i,j,k}+ \\
&\qquad\sum_{r=1}^{N}\gamma_{(r)}\sum_{r'=1}^{N'}\sum_{i=0}^{C_{in}-1}\sum_{j=0}^{K-1}\sum_{k=0}^{K-1}\frac{\boldsymbol{W^q}_{(r'),c,i,j,k}}{\beta_{(r')}} \\
&=\sum_{r=1}^{N}\sum_{r'=1}^{N'}\frac{1}{\alpha_{(r)}\beta_{(r')}}\boldsymbol{Y^q}_{(r,r'),c,h,w}+\sum_{r=1}^{N}\gamma_{(r)}\cdot\mathrm{sum}\left(\boldsymbol{W}_c\right) \quad (26)
\end{aligned}
$$

Therefore, let γ' be $\sum_{r=1}^{N}\gamma_{(r)}\cdot\mathrm{sum}\left(\boldsymbol{W}_c\right)$ as the new zero point, the dequantization for the output generated by convolutional layer based on residual quantization is calculated as follow:

$$
\boldsymbol{Y'}=\sum_{r=1}^{N}\sum_{r'=1}^{N'}\frac{1}{\alpha_{(r)}\beta_{(r')}}\boldsymbol{Y^q}_{(r,r')}+\gamma' \quad (27)
$$

IV. Restoration Experiment

A. Experiment Overview

In this experiment, we evaluate the effectiveness of output restoration. Fault generation, detection, and restoration simulations are performed on the software. For fault injection, we employ the Act Bit Flip method [7], which flips a single bit in the output feature maps with a specified probability. In restoration simulation, when a fault is detected, an approximated output based on either linear quantization or residual quantization is adopted. The iteration counts for residual quantization, N and N', are set to 2. We use a pre-trained ResNet-20 model as our evaluation model. Assuming an edge device implementation, all calculations in the original convolutional layers are performed with 8-bit precision. PyTorch 2.0.1 serves as the deep learning framework, and the model is further quantized to 8-bit precision using PyTorch's Quantization API. The layers in the 8-bit ResNet-20 model are converted to the proposed ATMR, with quantized bit widths m ranging from 3 to 7. The ATMR architecture, which consists of an 8-bit convolutional layer followed by two m-bit quantized convolutional layers, is denoted as ATMR(8,m,m).

Restoration is simulated using an 8-bit ResNet-20 model trained on the CIFAR-10 dataset, achieving an inference accuracy of 91.8%. The model's architecture is shown in Figure 2 (a). The first residual block in Layers 2 and 3 includes downsampling for the input, and its architecture is shown in Figure 2 (b). Other blocks are shown in Figure 2 (c). In this model, the area under the curves (AUCs) and thresholds corresponding to each layer are acquired from 10,000 images after setting FPR to 0.001. We select one convolutional layer from each layer group and the Stem layer, then evaluate the effects of faults and output restoration by injecting Act Bit Flip faults into the selected layers. Table I presents the changes in accuracy when Act Bit Flip faults occur with probabilities ranging from 1% to 20%. Based on the results, we identify the layers most affected by faults as targets for output restoration. Due to space limitations, in this paper, we present the results of output restoration for the selected layers ($stem.conv1$, $layer1.0.conv1$, $layer2.2.conv1$, $layer3.3.conv1$) where significant inference accuracy degradation is observed by fault injection.

B. Results

First, we present AUCs of fault detection of $stem.conv1$ in Table II, and the results of output restoration with linear quantization and residual quantization for $stem.conv1$ in Table III and Table IV, respectively. Both restorations of ATMR(8,3,3) cannot reach the original accuracy even when the AUC is 1, as $stem.conv1$ is sensitive to faults in terms of inference accuracy, and 3-bit residual quantization is insufficient for recovery given the low values of N and N'. With ATMR(8, 4, 4), the AUC reaches nearly 1 at a fault probability of 11%. The accuracy achieved through linear restoration is only about 2% lower than the original. Residual quantization achieves full recovery when the AUC is 1. This indicates that the bit width required for output restoration in ResNet-20 is at least 4 bits. Both restorations in ATMR(8,5,5) are less than a 10% reduction in inference accuracy since the AUC is 1 at 8%. The restoration by residual quantization performs slightly better for inference accuracy.

Second, we present AUCs for $layer1.0.conv1$ in Table V, along with the results of output restoration using linear quantization in Table VI and residual quantization in Table VII. For ATMR(8,3,3), restoration with residual quantization fully recovers the output when the AUC is 1 and performs better than linear quantization. From ATMR(8,4,4), both restorations have consistently maintained over 90%, and they are little difference.

Third, we present AUCs for $layer2.2.conv1$ in Table VIII, and the results of output restoration with linear quantization in Table IX and residual quantization in Table X. For ATMR(8,3,3), the AUCs reach 1 at fault probabilities where accuracy degradation becomes significant. Moreover, their TMRs show sufficient inference accuracy for both restorations.

Finally, we present AUCs for $layer3.1.conv1$ in Table XI, and the results of output restoration with linear quantization in Table XII and residual quantization in Table XIII. For ATMR(8,3,3), linear quantization provides better restoration accuracy than residual quantization, with accuracy degradation observed due to undetected faults when the AUC is not 1. From ATMR(8,4,4), both restorations show enough inference accuracy.

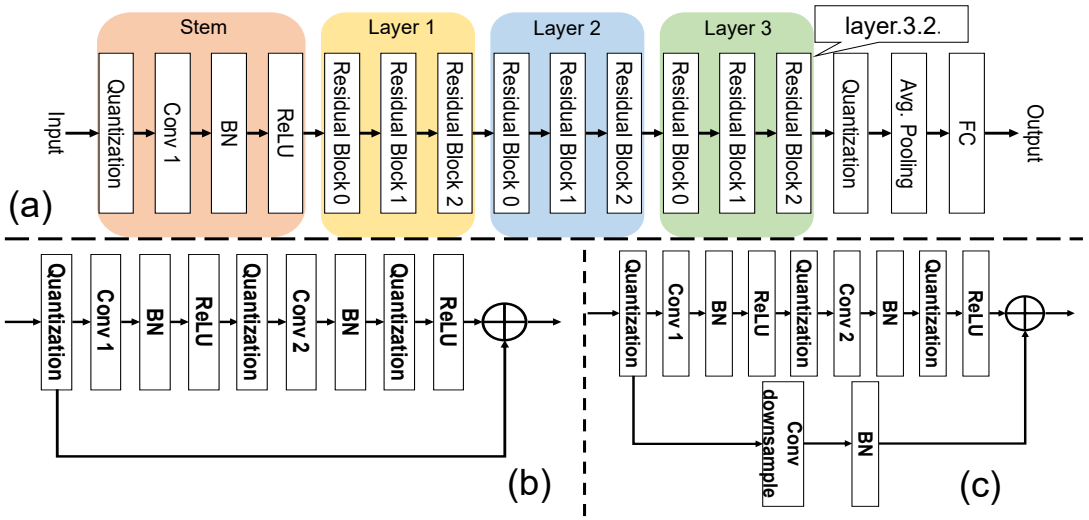

Fig. 2. ResNet-20 architecture.

TABLE I
ACCURACY CHANGING OF RESNET-20 BY ACT BIT FLIP.

Layer \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
stem.conv1	90.37	89.20	87.22	84.48	82.16	79.79	77.02	74.30	70.94	67.02	63.48	60.76	57.54	54.10	52.14	48.85	46.08	43.82	41.17	39.68
layer1.0.conv1	91.47	91.39	90.98	90.48	90.47	89.95	89.19	88.95	88.25	88.08	87.09	86.53	86.10	85.17	85.08	84.05	83.30	82.65	81.56	81.02
layer1.0.conv2	91.86	91.39	91.16	91.12	90.66	90.50	90.15	89.63	89.32	89.00	88.87	88.42	87.62	87.14	86.55	86.76	85.58	85.12	85.05	84.05
layer1.1.conv1	91.81	91.81	91.74	91.79	91.60	91.60	91.33	91.31	91.31	91.12	91.20	91.15	91.07	90.94	90.80	90.53	90.66	90.24	90.22	90.28
layer1.1.conv2	91.78	91.98	91.88	91.64	91.47	91.64	91.55	91.80	91.37	91.15	91.50	91.40	91.23	91.14	91.28	90.92	90.89	90.88	90.59	90.63
layer1.2.conv1	91.82	91.52	91.57	91.33	91.09	90.88	90.71	90.86	90.50	89.72	89.87	89.28	89.42	89.04	88.39	88.03	87.69	86.82	86.71	86.13
layer1.2.conv2	91.82	91.65	91.66	91.48	91.69	91.53	91.22	91.20	91.15	90.92	91.17	90.81	90.66	90.87	90.28	90.22	90.01	89.92	89.87	89.57
layer2.0.conv1	91.97	91.71	91.47	91.55	91.11	91.00	91.07	90.51	90.53	89.75	89.61	89.39	88.75	88.53	88.32	87.98	87.61	86.73	86.63	86.01
layer2.0.conv2	91.75	91.68	91.56	91.35	91.06	90.93	90.69	90.28	90.05	89.60	89.50	88.75	88.21	87.72	87.41	87.11	86.00	85.50	84.87	84.15
layer2.0.downsample.0	91.85	91.95	91.42	91.46	91.40	91.13	90.83	90.62	90.16	90.29	89.81	89.59	89.33	89.11	88.37	88.12	87.53	87.77	86.74	86.56
layer2.1.conv1	91.99	91.86	91.88	91.52	91.33	91.03	91.00	90.65	90.53	90.52	89.92	89.83	89.35	88.95	88.35	87.99	87.38	87.19	86.76	86.02
layer2.1.conv2	91.92	91.78	91.70	91.62	91.39	91.57	91.66	91.17	91.03	91.07	90.96	90.75	90.74	90.74	90.16	90.18	89.86	89.98	89.87	89.20
layer2.2.conv1	92.03	91.59	91.52	91.39	91.00	90.99	90.85	90.30	89.93	89.56	89.38	88.65	88.42	88.05	87.03	86.34	86.00	85.22	85.18	83.97
layer2.2.conv2	92.01	91.75	91.52	91.41	91.44	90.93	91.01	90.66	90.58	90.08	90.04	89.80	89.86	89.13	88.79	88.65	88.39	88.18	87.42	87.21
layer3.0.conv1	91.77	91.79	91.41	91.72	91.54	91.21	91.08	91.09	90.88	90.77	90.57	90.47	90.20	89.97	89.90	89.60	89.65	89.14	89.35	89.45
layer3.0.conv2	91.24	90.80	89.73	89.11	87.94	87.35	85.87	84.96	83.17	82.34	80.62	79.44	78.08	76.44	75.04	73.76	71.94	70.06	68.98	67.05
layer3.0.downsample.0	91.37	91.02	90.25	89.89	88.50	87.77	86.77	85.33	84.61	83.42	82.20	80.52	79.60	77.87	76.79	74.64	73.87	72.51	71.24	69.33
layer3.1.conv1	91.43	91.07	90.78	89.70	89.25	88.64	87.22	86.57	85.07	85.08	83.38	81.68	79.86	78.08	76.67	74.59	72.66	70.41	67.59	66.20
layer3.1.conv2	91.65	91.38	90.97	90.80	90.24	89.79	89.24	89.23	88.58	87.79	87.26	86.56	86.45	85.48	84.81	83.92	83.87	83.11	81.91	81.85
layer3.2.conv1	91.52	91.13	90.71	89.73	89.25	89.05	87.82	87.72	87.37	85.98	85.49	84.59	84.25	82.87	81.98	81.76	80.39	79.69	79.09	77.93
layer3.2.conv2	91.87	91.88	91.88	91.69	91.93	91.87	91.80	91.83	91.85	91.83	91.90	91.83	91.84	91.74	91.76	91.76	91.77	91.84	91.94	91.72

TABLE II
AUC OF STEM.CONV1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	0.576	0.650	0.718	0.778	0.830	0.872	0.906	0.932	0.951	0.966	0.976	0.984	0.989	0.993	0.995	0.997	0.998	0.999	0.999	1.000
(8,4,4)	0.663	0.796	0.886	0.939	0.969	0.984	0.992	0.996	0.998	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,5,5)	0.716	0.865	0.942	0.976	0.991	0.997	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,6,6)	0.825	0.948	0.985	0.996	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,7,7)	0.823	0.947	0.984	0.995	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

TABLE III
RESTORATION BY LINEAR QUANTIZATION FOR STEM.CONV1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	90.62	89.16	87.28	85.03	82.73	79.77	77.31	74.10	70.94	67.29	64.51	61.93	60.40	58.73	57.48	58.31	60.07	62.96	66.38	70.32
(8,4,4)	90.79	88.91	87.10	84.96	83.12	80.15	77.91	75.76	76.17	78.11	83.43	87.22	89.10	89.72	89.91	89.93	89.93	89.93	89.93	89.93
(8,5,5)	90.61	88.97	87.14	85.21	83.10	82.06	83.01	86.76	90.49	91.68	91.87	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91
(8,6,6)	90.65	88.98	87.36	85.42	86.41	90.13	91.75	92.09	92.12	92.12	92.12	92.12	92.12	92.12	92.12	92.12	92.12	92.12	92.12	92.12
(8,7,7)	90.64	89.54	87.14	85.64	85.24	89.56	91.65	92.20	92.25	92.25	92.25	92.25	92.25	92.25	92.25	92.25	92.25	92.25	92.25	92.25

TABLE IV
RESTORATION BY RESIDUAL QUANTIZATION FOR STEM.CONV1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	90.53	89.18	87.39	85.13	82.47	79.98	77.14	74.31	71.43	67.7	65.07	62.24	60.47	59.79	60.36	61.91	64.66	69.74	74.32	79.94
(8,4,4)	90.68	89.14	87.26	85.42	82.53	79.67	77.7	75.7	76.77	79.62	85.28	89.4	91.5	92.15	92.27	92.27	92.27	92.27	92.27	92.27
(8,5,5)	90.7	88.86	87.14	85.03	82.94	81.98	83.11	87.39	90.6	91.96	92.22	92.28	92.28	92.28	92.28	92.28	92.28	92.28	92.28	92.28
(8,6,6)	90.95	89.12	87.11	85.62	86.31	90.09	91.88	92.2	92.23	92.23	92.23	92.23	92.23	92.23	92.23	92.23	92.23	92.23	92.23	92.23
(8,7,7)	90.8	89.02	87.48	85.45	85.22	89.29	91.75	92.15	92.22	92.22	92.22	92.22	92.22	92.22	92.22	92.22	92.22	92.22	92.22	92.22

TABLE V
AUC OF LAYER1.0.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	0.708	0.854	0.932	0.969	0.986	0.993	0.997	0.998	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,4,4)	0.850	0.972	0.995	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,5,5)	0.990	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,6,6)	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,7,7)	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

TABLE VI
RESTORATION BY LINEAR QUANTIZATION FOR LAYER1.0.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	91.71	91.23	91.08	90.64	90.13	90.24	89.48	88.82	88.70	88.02	88.03	88.56	89.79	90.26	90.35	90.35	90.35	90.35	90.35	90.35
(8,4,4)	91.70	91.08	91.04	90.71	90.48	91.23	91.68	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82
(8,5,5)	91.66	91.73	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86
(8,6,6)	91.81	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99	91.99
(8,7,7)	91.86	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85	91.85

TABLE VII
RESTORATION BY RESIDUAL QUANTIZATION FOR LAYER1.0.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	91.72	91.40	90.84	90.84	90.08	89.99	89.12	89.08	88.32	88.54	88.28	89.41	91.10	91.71	91.80	91.80	91.80	91.80	91.80	91.80
(8,4,4)	91.67	91.31	91.23	90.55	90.56	90.79	91.85	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86
(8,5,5)	91.87	91.86	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88	91.88
(8,6,6)	91.93	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90
(8,7,7)	91.96	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00

TABLE VIII
AUC OF LAYER2.2.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	0.680	0.823	0.916	0.966	0.988	0.997	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,4,4)	0.859	0.984	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,5,5)	0.988	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,6,6)	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,7,7)	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

TABLE IX
RESTORATION BY LINEAR QUANTIZATION FOR LAYER2.2.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	91.77	91.51	91.46	91.04	91.49	91.06	91.23	91.30	91.37	91.38	91.38	91.38	91.38	91.38	91.38	91.38	91.38	91.38	91.38	91.38
(8,4,4)	91.73	91.70	91.72	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74	91.74
(8,5,5)	91.80	91.92	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93	91.93
(8,6,6)	91.93	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89
(8,7,7)	91.82	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92	91.92

TABLE X
RESTORATION BY RESIDUAL QUANTIZATION FOR LAYER2.2.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	91.76	91.73	91.30	91.15	91.10	91.52	91.62	91.79	91.85	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84
(8,4,4)	91.83	91.79	91.82	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91	91.91
(8,5,5)	91.83	91.77	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78
(8,6,6)	91.96	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95
(8,7,7)	91.90	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89

TABLE XI
AUC OF LAYER3.1.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	0.664	0.796	0.885	0.937	0.967	0.984	0.993	0.997	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,4,4)	0.838	0.960	0.992	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,5,5)	0.957	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,6,6)	0.996	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
(8,7,7)	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

TABLE XII
RESTORATION BY LINEAR QUANTIZATION FOR LAYER3.1.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	91.51	91.19	90.70	89.77	89.42	88.61	88.70	88.76	89.71	90.53	90.96	91.16	91.19	91.20	91.20	91.20	91.20	91.20	91.20	91.20
(8,4,4)	91.36	91.05	91.10	91.68	91.82	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90	91.90
(8,5,5)	91.77	91.82	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95	91.95
(8,6,6)	91.80	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86	91.86
(8,7,7)	91.91	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89	91.89

TABLE XIII
RESTORATION BY RESIDUAL QUANTIZATION FOR LAYER3.1.1 IN RESNET-20.

ATMR \ Fault_prob[%]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(8,3,3)	91.79	91.03	90.04	89.54	89.00	87.95	87.58	87.78	88.08	89.10	89.28	89.55	89.56	89.58	89.58	89.58	89.58	89.58	89.58	89.58
(8,4,4)	91.38	91.00	90.84	91.31	91.57	91.58	91.60	91.60	91.60	91.60	91.60	91.60	91.60	91.60	91.60	91.60	91.60	91.60	91.60	91.60
(8,5,5))	91.58	91.78	91.84	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82	91.82
(8,6,6)	91.74	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78	91.78
(8,7,7)	91.79	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84	91.84

V. DISCUSSION

From the evaluation of ResNet-20, ATMR of $stem.conv1$ cannot effectively handle faults without high bit quantization due to missing errors. In contrast, other layers can achieve near-original accuracy recovery even with 3- or 4-bit quantization. The $stem.conv1$ is the convolution layer with the fewest number of filters, and other layers that are placed later gradually increase the number of filters. Therefore, large circuit resources and power savings are expected when all TMR units of ResNet-20 except $stem.conv1$ are replaced with ATMR units with 3 or 4 bits. Moreover, using residual quantization for fault detection in $stem.conv1$ has the potential to improve the AUC with 3- or 4-bit quantization.

VI. CONCLUSION

In this paper, we propose an ATMR system capable of detecting faults and continuing inference with output restoration, resulting in reduced circuit area and power consumption compared to conventional TMR. Additionally, we propose using output restoration methods based on linear quantization and residual quantization. For evaluation, we conducted experiments that were a simulation of output restoration using ResNet-20 in software. Results from ResNet-20 showed that 3- or 4-bit restoration was effective in layers except for $stem.conv1$. In our future work, we plan to implement ATMR circuits on an FPGA to measure resource utilization and power consumption.

REFERENCES

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research, Volume 18, Issue 1, pp 6869–6898*, 2016.

[4] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *Neural Information Processing Systems (NIPS)*, 2018.

[5] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," *International Conference on Learning Representations (ICLR)*, 2017.

[6] E. Maricau and G. Gielen, "Transistor aging-induced degradation of analog circuits: Impact analysis and design guidelines," *European Solid State Circuits Conference (ESSCIRC)*, 2011.

[7] Y. Saikawa, Y. Owada, Y. Tomioka, H. Saito, and Y. Kohira, "Fault detectable convolutional neural network circuits with dual modular redundancy based on mixed-precision quantization," in *IEICE Technical Report (VLD2023-122)*, 02 2024, pp. 99–140, (In Japanese).

[8] H. Xu, Z. Chen, W. Wu, Z. Jin, S.-y. Kuo, and M. Lyu, "Nv-dnn: Towards fault-tolerant dnn systems with n-version programming," *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2019.

[9] S. Salamin, G. Zervakis, O. Spantidi, I. Anagnostopoulos, J. Henkel, and H. Amrouch, "Reliability-aware quantization for anti-aging npus," in *2021 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2021, pp. 1460–1465.

[10] Y. Li, Y. Liu, M. Li, Y. Tian, B. Luo, and Q. Xu, "D2nn: a fine-grained dual modular redundancy framework for deep neural networks," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 138–147.

[11] Y. Owada, Y. Tomioka, and H. Saito, "Dual modular redundancy unit of convolutional layer for low-cost and reliable cnns," in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2022, pp. 379–384.

[12] E. Yvinec, A. Dapgony, M. Cord, and K. Bailly, "Rex: Data-free residual quantization error expansion," 2023. [Online]. Available: https://arxiv.org/abs/2203.14645