# ExtendLife: Weights Mapping Framework to Improve RRAM Lifetime for Accelerating CNN

Fan Yang[1], Yusen Li[1(✉)], Zeyuan Niu[2], Gang Wang[1], and Xiaoguang Liu[1]

[1] College of Computer Science, Nankai University, Tianjin, China
{yangf,liyusen,wgzwp,liuxg}@nbjl.nankai.edu.cn
[2] College of Information Engineering, Southeast University, Nanjing, China

**Abstract.** Process-in-memory (PIM) engines based on Resistive random-access memory (RRAM) are used to accelerate the convolutional neural network (CNN). RRAM performs computation by mapping weights on its crossbars and applying a high voltage to get results. The computing process degrades RRAM from the fresh status where RRAM can support high data precision to the aged status where RRAM only can support low precision, potentially leading to a significant CNN training accuracy degradation. Fortunately, many previous studies show that the impact of loss caused by the RRAM precision limitation across various weights is different for CNN training accuracy, which motivates us to consider mapping different weights on RRAM with different statuses to keep high CNN training accuracy and extending the high CNN training accuracy iterations of PIM engines based on RRAM, which is regarded as the lifetime of the RRAM on CNN training. In this paper, we propose a method to evaluate the performance of the weights mapping on extending the lifetime of the RRAM and present a weights mapping framework specifically designed for the hybrid of aged and fresh RRAM to extend the lifetime of the RRAM engines on CNN training. Experimental results demonstrate that our weights mapping framework brings up to 6.3× on average lifetime enhancement compared to the random weights mapping.

**Keywords:** RRAM · CNN · Weights Mapping · Process-In-Memory

## 1 Introduction

Convolutional neural networks have gained remarkable performances in various fields such as facial recognition [1], image classification [2], and pattern recognition [3]. CNN can usually be deployed on different devices to support the corresponding service. Since the low power consumption of RRAM and its ability to calculate matrix multiplication quickly, PIM engines based on RRAM are often used to deploy CNN [4]. Compared with the traditional von-Neumann architectures, e.g., CPUs and GPUs, PIM engines based on RRAM can efficiently perform amounts of vector-matrix multiplications by using RRAM crossbars


C. Li et al. (Eds.): APPT 2023, LNCS 14103, pp. 40–53, 2024.
https://doi.org/10.1007/978-981-99-7872-4_3

as shown in Fig. 1 to implement these computations, where the matrix data $G$ is mapped onto RRAM and the voltage vector $V$ representing input vector is applied to the crossbar, and then each column of RRAM generates current indicating results according to Kirchoff's Law.

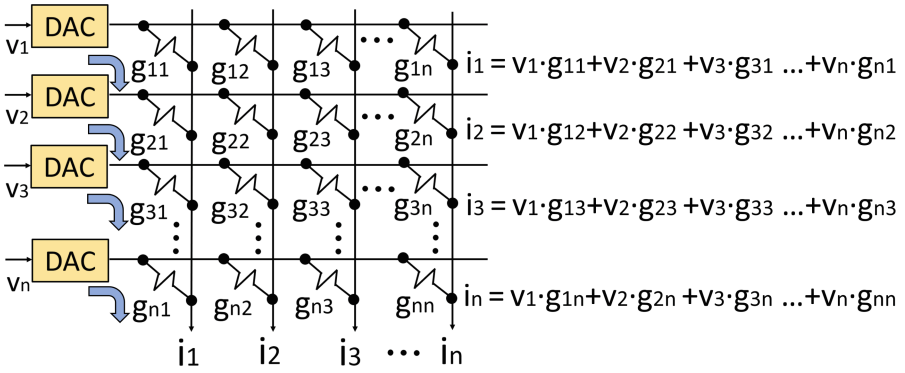


**Fig. 1.** The RRAM crossbar computation progress

Fig. 1 illustrates how to perform the vector-matrix multiplication on the RRAM crossbar. As Fig. 1 shows, $g_{11}$ means the RRAM unit in the first row and the first column of the RRAM crossbar, and the $g_{11}$ value presents the conductance of the RRAM unit, which can be programmed according to the data on the specific location of the matrix so that all the matrix data can be mapped onto the RRAM crossbar. After mapping matrix data on the RRAM crossbar, we need to apply the input vector to the RRAM crossbar to finish the vector-matrix multiplication. On the RRAM crossbar, the input vector is programmed to the voltage vector $V = [v_1, v_2, v_3, ..., v_n]$, where $v_1$ is applied to the first row in the crossbar.

According to Kirchoff's Law, the output current value of each column on the RRAM crossbar equals the dot product result of the input voltage vector $V$ and the vector consisting of the RRAM units conductance in the corresponding column. For example, the first column current is $i_1 = v_1 \cdot g_{11} + v_2 \cdot g_{21} + v_3 \cdot g_{31} + ... + v_n \cdot g_{n1}$. With the crossbar computing process above, the RRAM crossbar can be used to efficiently perform vector-matrix multiplications within a few cycles, which is crucial for accelerating CNN. And the crossbar architecture has been widely integrated into different accelerators such as ISAAC [5], Prime [6], and Timely [7].

To train CNN on the RRAM crossbar, a common method is online tuning, which means the conductance of the RRAM units is firstly programmed according to the initialized weights and then tuned in further iterations. During online tuning, RRAM is reprogrammed by applying the high voltage many times, which causes subtle changes in RRAM units, resulting in a decrease in the data precision that RRAM supports, which is called aging. RRAM units that are never

used are in the fresh status, where they can support the highest data precision, and ones that have been programmed many times are in the aged status, where they cannot support high data precision. If the precision of the mapped data exceeds what the RRAM units support, this may result in the data loss of the mapped data, that ultimately influences the high CNN training accuracy iterations, which is regarded as the lifetime of the RRAM on training CNN.

In our previous surveys, we find that many factors affect RRAM lifetime, e.g., temperature [8] and noise [9]. Many methods [10] have been proposed to extend RRAM lifetime from temperature and noise views. But we don't find methods proposed to extend the RRAM lifetime from the perspective of weights mapping. Although the CNN workload balance [11] can reduce the status gap among the different RRAM units to some extent, there also is left space for improvement. We find that the Network Pruning theories [12] can be applied to this problem, in these theories, some weights that hardly affect network accuracy can be removed, which reminds us to consider that the weights mapping framework can be performed on the hybrid of the fresh and aged RRAM to extend RRAM lifetime.

In this paper, we evaluate the performance of the weights mapping on RRAM lifetime based on Taylor expansion. According to our analysis, we propose the weights mapping framework to improve RRAM lifetime in the hybrid environment of aged and fresh RRAM. Our contributions are:

- We propose an evaluating method based on Taylor expansion to compare two opposite weights mappings performance.
- According to our evaluating method, we propose the weights mapping framework consisting of two methods: 1. Weights mapping based on the layer position in CNN, 2. Weights mapping based on the characteristic of the groups which split weights in the same kernel.
- We do experiments about three VGG-like networks as shown in Table 1 (NN5, NN7, NN9) on two datasets (MNIST, Fashion). Compared with the random weights mapping, our weights mapping framework can improve the RRAM lifetime by 6.3× on average.

## 2 Background and Motivation

In this section, we will review the CNN training on RRAM crossbars, RRAM aging, and other related backgrounds.

### 2.1 CNN Training on RRAM

The convolution operator is the most common operator in CNN, which extracts information from the input data by applying the convolution kernel. Figure 3 depicts the kernel weights $W$ mapping on RRAM, where the convolution kernel, with 3 channels, convolutes on the input data $X$. The weights $W$ within each channel are converted into a one-dimensional vector, which is mapped on a

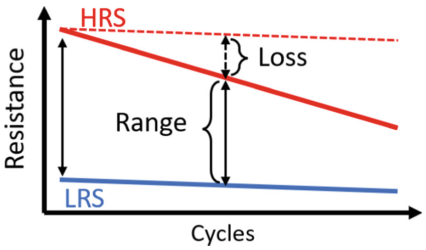


**Fig. 2.** The aging progress of RRAM

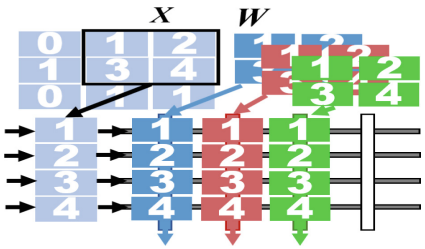


**Fig. 3.** Mapping the convolution kernel weights on RRAM

column of the crossbar, and then, as Fig. 1 shown, the input data is also converted to a one-dimensional voltage vector and this voltage vector is applied to the crossbar to generate the results.

After $W$ is mapped on the RRAM crossbar, the output of the network is compared with the expected output in the training data, and the difference between them is defined as the $Cost$, which is used as the feedback to the neural network to adjust the weights. $\mathcal{LR}$ denotes the learning rate. According to $Cost$, the backpropagation is then applied to update the weights as below equation in CNN to reduce the $Cost$ for higher training accuracy.

$$W = W - \mathcal{LR} \cdot \frac{\partial Cost}{\partial W} \tag{1}$$

### 2.2 RRAM Aging

Both Mapping weights on RRAM and adjusting the weights need applying the high voltage to RRAM, inducing tiny changes within the RRAM cells [13]. Regrettably, these tiny changes are irreparable, leading to RRAM aging and a permanent decrease in its supporting precision. The phenomenon of RRAM aging is depicted in Fig. 2, where HRS and LRS denote the upper and lower resistance of the RRAM. The interval between HRS and LRS denotes the range that RRAM uses to support data precision. The latest single RRAM cell can store 3 bits and the byte is usually represented by a fixed number RRAM units [14]. As the times of RRAM applied to high voltage increases, tiny changes accumulate in the RRAM cells, and finally, the accumulated changes decrease the precision that RRAM supports.

### 2.3 Weights Distribution and Network Pruning

To avoid gradient disappearance and gradient explosion, many initialization methods such as Kaiming initialization have been integrated into PyTorch [15]. These initialization methods normalize the distribution of weights across each layer of CNN, and prior research has demonstrated that the trained weights

mostly conform to a Gaussian-alike distribution [16], which reminds me to explore the influence of weights mapped on the different statuses RRAM units on CNN training.

Furthermore, we need to determine how the loss of different weights on aged RRAM affects network training. In this regard, our problem is very similar to Network Pruning, but with two differences: 1. Network Pruning often removes useless weights, while RRAM reduces the value of weight to the lower one because of the RRAM status limitation, and 2. Network Pruning methods usually require complete network training and repeat fine-tuning, and these methods need much computation and may further age RRAM. Additionally, as RRAM computes based on the crossbars, many Network Pruning methods like unstructured pruning are also difficult to adapt to RRAM.

In the following sections, we discuss the evaluating methods of the two opposite weights mapping methods and propose the weights mapping framework suitable for RRAM crossbars and our framework achieves better performance compared to the random weights mapping baseline.

## 3 Evaluate Weights Mapping

To map different weights to RRAM with different statuses in a reasonable way, we want to know how to evaluate the weights mapping to some extent in advance. Consider a set of training examples $\mathcal{D} = \{\mathcal{X} = \{x_0, x_1, \ldots, x_n\}, \mathcal{Y} = \{y_0, y_1, \ldots, y_n\}\}$, where $x_i$,$y_i$ respectively represent an input and a target output. The weights $\mathcal{W} = \{w_0, w_1, \ldots, w_n\}$ are optimized to minimize the cost value $Cost(\mathcal{D}, \mathcal{W})$.

We find the weights update magnitude from backpropagation usually drops out of the range that aged RRAM supports. Because the update of the weights in the layer follows $\frac{\partial Cost}{\partial w^l} = \delta^l \cdot \sigma\left(z^{l-1}\right)$, where $\sigma\left(z^{l-1}\right)$ is the previous layer output and $\delta^l$ means the derivative of $Cost$ to $z^l$. To avoid gradient disappearance and gradient explosion, each layer output needs to be normalized, causing the absolute value of $\sigma\left(z^{l-1}\right)$ and $\delta^l$ is less than 1.0. So the update magnitude on weights following Eq. 1 is less than $\mathcal{LR}$. $\mathcal{LR}$ is usually related to iteration speed and overfitting, to take a good compromise, it's common to pick 0.0001 as $\mathcal{LR}$. This means the update magnitude on weights is usually less than 0.0001. The aged RRAM can not support such high precision, so the weights in CNN training on the aged RRAM are not precisely updated, and eventually, the effects of backpropagation will be useless on the aged RRAM.

In this situation, we evaluate the weights mapping on the training network based on Taylor expansion. As $|\Delta Cost| = |\mathcal{C}(\mathcal{D}, \mathcal{W}') - \mathcal{C}(\mathcal{D}, \mathcal{W})|$ shows, where $\mathcal{W}$ means the original weights and $\mathcal{W}'$ is the weights mapped on the RRAM with different statuses, $|\Delta Cost|$ means the difference caused by weights change because of the RRAM statuses limitation.

For notational convenience, we temporarily consider the weights mapped on the aged RRAM as $\alpha W$, where $\alpha$ is an aged factor ranging from 0.0 to 1.0. The

larger $\alpha$ indicates the weights mapped on the fresher RRAM. Then, the $|\Delta Cost|$ is can be expressed as $|\Delta Cost| = |\mathcal{C}(\mathcal{D}, W) - \mathcal{C}(\mathcal{D}, \alpha W)|$.

To approximate $|\Delta Cost|$, we use the first-degree Taylor polynomial. For a function $f(x)$, the Taylor expansion at point $x = a$ is $f(x) = \sum_{p=0}^{P} \frac{f^{(p)}(a)}{p!}(x - a)^p + R_p(x)$, where $f^{(p)}(a)$ is the $p$-th derivative of $f$ evaluated at point $a$, and $R_p(x)$ is the $p$-th order remainder. Approximating $\mathcal{C}(\mathcal{D}, \alpha W)$ with a first-order Taylor polynomial near $\alpha W$, we have: $\mathcal{C}(\mathcal{D}, \alpha W) = \mathcal{C}(\mathcal{D}, W) - \frac{\partial \mathcal{C}}{\partial W}(W - \alpha W) + R_1(W = \alpha W)$. The remainder $R_1(W = \alpha W)$ can be neglected largely because of the significant calculation required, but also in part because the widely-used ReLU activation function encourages a smaller second order term [17], so we substitute the above equations, we have: $|\Delta Cost| = |\mathcal{C}(\mathcal{D}, W) - \mathcal{C}(\mathcal{D}, \alpha W)| = \left|\frac{\partial \mathcal{C}}{\partial W}(W - \alpha W)\right| = \left|(1 - \alpha)\frac{\partial \mathcal{C}}{\partial W} W\right|$. $\frac{\partial \mathcal{C}}{\partial W}$ usually needs enormous amounts of computation, many studies [18] choose to neglect its effect temporarily. Intuitively, $|\Delta Cost|$ is much related to $(1 - \alpha)$ and $W$. It shows that the RRAM aged factor and the weights themselves can affect $|\Delta Cost|$. Based on that, we can infer that we should avoid mapping the weights with the high value on the aged RRAM.

For this, we conduct experiments to compare the opposite weights mapping methods in advance, which are illustrated in detail in the next section. We assume that we map weights on the RRAM which has half aged units and half fresh units. In Fig. 4, we accumulate the loss between the original value and the value mapped on the different statuses RRAM units to get information loss, and *back-front* means that we map the weights of the front/back half of the network on the aged RRAM and map the left half weights on the fresh RRAM. We can observe that the *back* has less information loss than *front* in NN5, NN7 and more information loss in NN9, these networks structure will be illustrated in Table 1. According to the information loss, we can predict the *back* performs better compared with *front* in NN5, NN7, and has worse performance than *front* in NN9.

We also compare the information loss between *over* and *below* weights mapping, which groups the weights within the same kernel and maps the groups on the different statuses RRAM according to the absolute sum of the groups. Based on Fig. 4, we find that *below* has less information loss than *over* and we predict *below* has better performance than *over*. The final experiments results show that our evaluation between the opposite weights mapping methods is effective.

## 4 Weights Mapping Framework

According to the comparison of information loss in the above section, we propose two weights mapping methods. The first is to map all the weights of different convolutional kernels to the RRAM with different statuses based on the layer position of weights in the network. The second is to group the weights of each convolution kernel for RRAM crossbar computation and map them based on the values of the grouped weights.

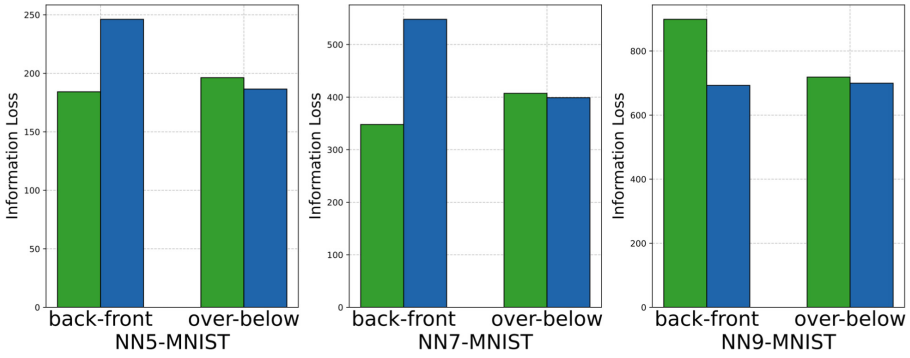


**Fig. 4.** Compare information loss of different weights mappings. The bar means the information loss of weights mapping.

### 4.1 Dataflow

As shown in Fig. 5, there are three RRAM crossbars with varying degrees of aged, labeled as Fresh, Old and Older, with Fresh exhibiting the minimum degree of aged, followed by Old, and Older. In training CNN, the weights in the convolution kernels need to be mapped on the RRAM crossbar. There are two simple mapping directions in Fig. 5: Dataflow1 maps the Conv1 weights on the Fresh crossbar, the Conv2 weights on the Old crossbar, and the Conv3 weights on the Older crossbar, while Dataflow2 sets the Conv1 weights to the Older crossbar, the Conv2 weights to the Old crossbar, and the Conv3 weights to the Fresh crossbar, with the input data streams going through Conv1 followed by Conv2 and Conv3.

In the real case, our usual choice is Dataflow2. The weights in the back half of the network are mapped on the aged RRAM and the front half weights are mapped on the fresh RRAM, this weights mapping, called *back*, results in less information loss compared with the opposite weights mapping in some networks as shown in Fig. 4.

### 4.2 Grouping and Mapping Groups

In the above section, we observe that the approach of mapping weights to RRAM with different statuses based on the layer position may become ineffective as CNN becomes deep as shown in Fig. 4. This challenge leads us to explore a weights mapping approach that applies to all deep CNN for RRAM crossbars.

As illustrated in Fig. 6, we extract the kernel weights in the stripe-wise direction and map them to the column in the RRAM crossbar. Similarly, we extract input data in the same direction to apply it on the crossbar. Each crossbar generates a part of this layer output, and all the results from the crossbars must be summed on the position of the next layer. As shown in Fig. 6, $K \times K$ crossbars are required for computation, and $K \times K$ outputs must be aggregated to derive the output at the intended position of the next layer.

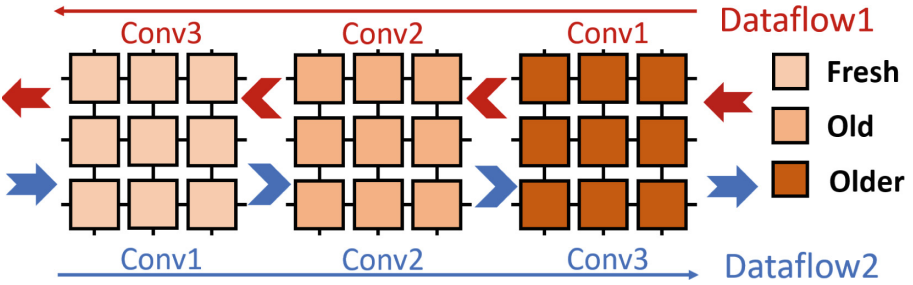


**Fig. 5.** Two dataflows: Dataflow1 maps the front half of weights in the network on the aged RRAM, and Dataflow2 maps the back half of weights in the network on the aged RRAM.

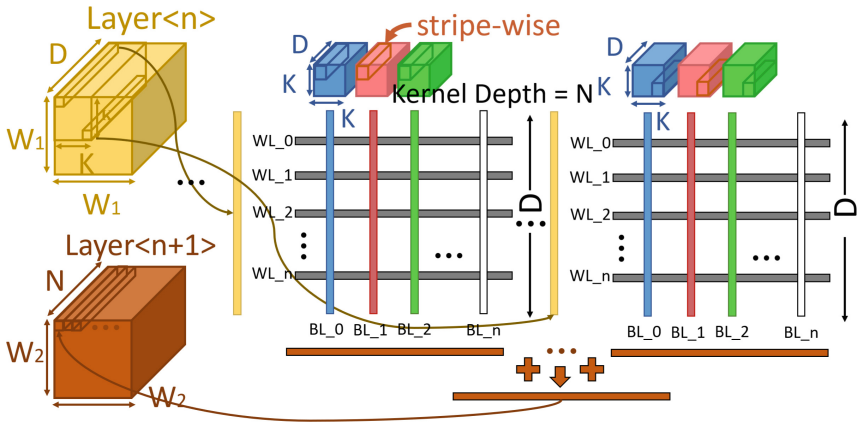


**Fig. 6.** We group weights in the stripe-wise direction and the input data that is extracted in the stripe-wise direction is applied to the crossbar.

Compared to the traditional weights mapping shown in Fig. 3, the grouping in the stripe-wise direction brings significant advantages. Since the traditional mapping method converts the weights of each channel into a one-dimensional vector and maps it to the column of the RRAM crossbar, and this method requires a large RRAM crossbar with the size of $N \times (K \times K \times D)$. Our mapping as Fig. 6 shows groups the weights in the stripe-wise direction and maps the groups to many little size RRAM crossbars with different statuses, and avoid using the one whole large size RRAM crossbar consisting of different statuses RRAM units.

**Algorithm 1:** The weights mapping on crossbars

1 **Input:** The weights in one kernel are grouped in the stripe-wise direction $W = \{w_1, w_2, ..., w_n\}$, $w_i$ means the weights set in $i$-th group.
2 **for** *each* $w_i$ *in* $W$ **do** // Traverse all $w_i$
3 Compute the sum of the absolute value of $w_i$ $S_i = sum(\|w_i\|)$
4 Order $S_i$ in descending order and map the groups with the greater $S_i$ on the fresh RRAM, and map the other half groups with the lower $S_i$ on the aged RRAM.

From Fig. 4, we know that the weights information loss caused by RRAM with different statuses has different impacts on CNN. As shown in Fig. 7, after grouping the weights, we map the weights onto the crossbars with different statuses as Algorithm 1. We will aggregate the information of each group, sort it according to the absolute value sum of the group, and then map the grouped weights to the different statuses crossbars.

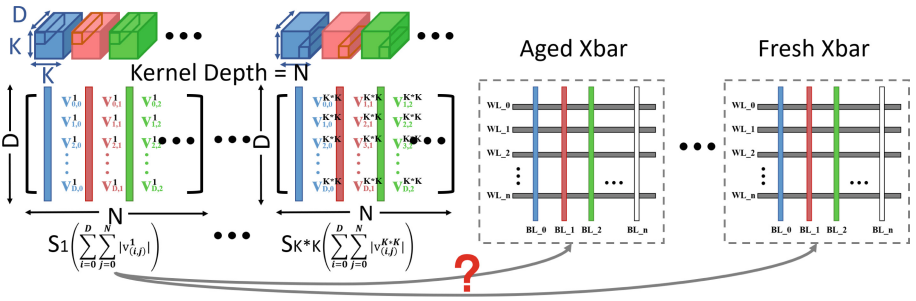


**Fig. 7.** After we group the weights, we map the groups on the crossbars with different statuses as Algorithm 1 describes.

## 5 Experiments

To evaluate the performance of our weights mapping approaches, we test three different neural networks, NN5, NN7, and NN9, on two datasets, MNIST and Fashion [19]. We use three VGG-like networks shown in Table 1, where NN5 has 5 convolution layers, NN7 has 7 convolution layers, and NN9 has 9 convolution layers to explore the effect of different weights mapping on networks with different depths. Because the simulation continuously changes the weights according to the RRAM units' statuses, the simulation will cost much time even though we test small datasets such as MNIST and Fashion at first. MNIST and Fashion both contain 60,000 images, and the images are divided into 10 classes. We conduct simulation experiments in the hybrid environment of aged and fresh RRAM,

where the number of fresh RRAM units is the same as the aged RRAM ones. All network implementations and the weights mapping framework are implemented using Numpy [20]. The experiments are tested on an Intel Xeon Silver 4114 CPU @ 2.20GHz.

**Table 1.** CNN Configuration

| CNN Configuration | | |
|---|---|---|
| NN5 | NN7 | NN9 |
| 5 conv layers | 7 conv layers | 9 conv layers |
| input ($28 \times 28$ image ) | | |
| conv3-16 | conv3-16 | conv3-16 |
| maxpool | | |
| conv3-16 | conv3-16 | conv3-16 |
| maxpool | | |
| conv3-16 | conv3-16 | conv3-16 |
| conv3-32 | conv3-32 | conv3-32 |
| conv3-64 | conv3-32 | conv3-32 |
| | conv3-64 | conv3-32 |
| | conv3-64 | conv3-64 |
| | | conv3-64 |
| | | conv3-64 |
| FC-128 | | |
| FC-10 | | |
| soft-max | | |

Table 2 shows the performance of different weights mapping methods on different datasets and networks, where the data represents the RRAM lifetime improvement compared to the random weights mapping, which maps the weights on the same RRAM randomly. The lifetime is the iteration when the CNN training batch accuracy drops down the borderline, which usually is defined as 80% of the original final accuracy from the test data [21], and we use the same rule. The crosses in the next figures mark the lifetime of the RRAM. The *front-half* means we map the weights of the front half of CNN on the aged RRAM and map the other weights on the fresh RRAM, and the *back-half* is opposite to the *front-half*. It can be seen that the *front-half* weights mapping is inferior to the random weights mapping in all networks, and does not exhibit significant changes on the networks of different depths.

Unlike the *front-half*, the *back-half* maps the weights of the back half of the CNN to the aged RRAM, and maps the front half to the fresh RRAM. Table 2 shows that the back-half significantly extends 4.9× lifetime in NN7. Although

the performance of the back-half on NN5 is not outstanding, Fig. 8 shows that *back-half* still extends the lifetime around 0.6 accuracy in NN5. Comparing Fig. 9 and Fig. 10, we can see that the lifetime improvement of the *back-half* becomes less significant with the depth of CNN increasing, indicating that the simple front or back half weights mapping on RRAM may become ineffective in the deep CNN.

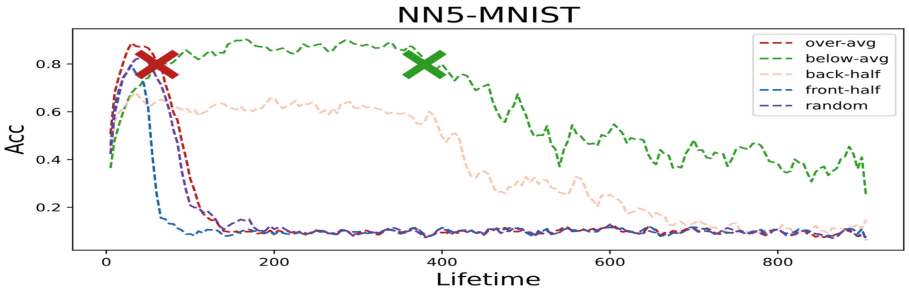


**Fig. 8.** NN5-MNIST lifetime improvement

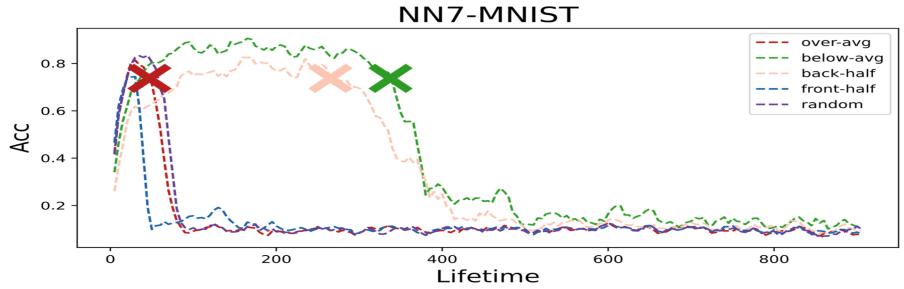


**Fig. 9.** NN7-MNIST lifetime improvement

Interestingly, we find the results from Table 2 are different from Fig. 4 in NN7 and NN9. But if we lower the borderline determining the lifetime, we can easily find that in NN5 and NN7, *back-half* has better performance, and in NN9 it has almost worse performance than *front-half* from Fig. 8, Fig. 9 and Fig. 10 which corresponds to the Fig. 4.

For solving the issue of possible inefficiency of *back-half* in deep CNN, we also propose a mapping method that groups the weights and maps them to the crossbars with different aged statuses. The main criterion for this mapping is the absolute sum of grouped weights as Algorithm 1 describes. The *over-avg* means that we map the half groups that have the greater absolute sum on the aged RRAM, and map the other half groups on the fresh RRAM. As shown in

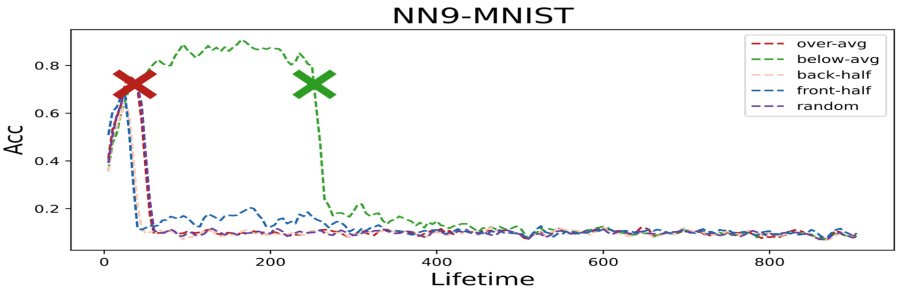


**Fig. 10.** NN9-MNIST lifetime improvement

Table 2, the *over-avg* shows a similar speedup to the random weights mapping among all the networks.

The *below-avg* is opposite to the *over-avg*, *below-avg* maps the half groups with less absolute value sum to the aged crossbar and maps the other half groups on the fresh crossbar. As shown in Table 2, the *below-avg* demonstrates a significant and stable lifetime improvement across all networks. This highlights the meaningfulness of weights grouping in the convolution kernel, which not only allows a more flexible and friendly mapping for the RRAM crossbars but also significantly enhances lifetime.

**Table 2.** Weights mapping lifetime improvment

| NN | Dataset | back-half | front-half | over-avg | below-avg |
|---|---|---|---|---|---|
| NN5 | MNIST | 0.8× | 0.8× | 1.3× | 7.1× |
| | Fashion | 1.1× | 0.7× | 0.7× | 5.5× |
| NN7 | MNIST | 4.8× | 0.5× | 0.9× | 5.6× |
| | Fashion | 5.0× | 0.7× | 1.0× | 7.2× |
| NN9 | MNIST | 0.8× | 0.6× | 0.9× | 5.7× |
| | Fashion | 1.0× | 0.7× | 1.3× | 7.2× |

Although Fig. 4 shows the loss of the four weights mapping, the *back/front-half* loss is difficult to be compared with the *below/over-avg* one. Because the loss between *back-half* and *front-half* is primarily caused by the weights distribution difference in the different layers. But the loss between *below-half* and *over-half* generates from the same kernel, its difference is dramatically smaller than the loss between *back-half* and *front-half*. Even though the information loss cannot be compared with the weights mapping *back/front-half* based on the dataflow and the one *below/over-avg* within the kernel directly, it can be used to compare the weights mappings based on the same way such as *back-half* and *front-half* or *below-half* and *over-half*. In the above figures, the accuracy drops very fast

in some intervals, the phenomenon is similar to *layer-collapse* in Pruning Networks [22], which generates from the weights loss caused by the aged RRAM units and can be hardly avoided.

## 6 Acknowledgment

This work is supported by Key-Area Research and Development Program of Guangdong Province 2021B0101310002; National Science Foundation of China (grant numbers 62293510/62293513, 62272252, 62272253, 62141412), NSF of Tianjin 21JCYBJC00070; Fundamental Research Funds for the Central Universities.

## 7 Conclusions

In this paper, we analyze the impact of aged RRAM on training CNN and proposed a method to evaluate the performance of weights mapping on the hybrid of aged and fresh RRAM through Taylor expansion in advance. Based on this method, we propose a weights mapping framework specifically designed for the hybrid of aged and fresh RRAM to extend RRAM engines lifetime consisting of two weights mapping methods. The *back-half* maps the weights of the back half of the network to aged RRAM, and the *below-avg* groups the weights and maps the groups with smaller absolute value sum to aged RRAM. Experimental results show that both *back-half* and *below-avg* can improve the lifetime of RRAM accelerators. However, in experiments, *back-half* varies in lifetime improvement in CNN with different depths, while *below-avg* can stably improve the RRAM lifetime in all CNN. Compared with random weights mapping baseline, *below-avg* can improve the RRAM lifetime by 6.3× on average.

## References

1. Cokun, M., et al.: Face recognition based on convolutional neural network. In: International Conference on Modern Electrical and Energy Systems (2017)
2. Li, Q., et al.: Medical image classification with convolutional neural network. In: 13th International Conference on Control Automation Robotics & Vision (2014)
3. Chen, L., et al.: Beyond human recognition: a CNN-based framework for handwritten character recognition. In: 3rd IAPR Asian Conference on Pattern Recognition (2015)
4. Tang, T., et al.: Binary convolutional neural network on RRAM. In: 22nd Asia and South Pacific Design Automation Conference (2017)
5. Shafiee, A., et al.: ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In: ACM SIGARCH Computer Architecture News, vol. 44, no. 3, pp. 14–26 (2016)
6. Chi, P., et al.: Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory. In: ACM SIGARCH Computer Architecture News, vol. 44, no. 3, pp. 27–39 (2016)

7. Li, W., et al.: Timely: pushing data movements and interfaces in pim accelerators towards local and in time domain. In: ACM/IEEE 47th Annual International Symposium on Computer Architecture (2020)
8. Chen, P.Y., et al.: WRAP: weight RemApping and processing in RRAM-based neural network accelerators considering thermal effect. In: Design, Automation & Test in Europe Conference & Exhibition (2022)
9. Zhang, G.L., et al.: Reliable and robust RRAM-based neuromorphic computing. In Proceedings of the 2020 on Great Lakes Symposium on VLSI (2020)
10. Zhang, S., et al.: Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects. In: 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (2020)
11. Zhu, Z., et al.: Mixed size crossbar based RRAM CNN accelerator with overlapped mapping method. In: International Conference on Computer-Aided Design (2018)
12. Liang, T., et al.: Pruning and quantization for deep neural network acceleration: a survey. Neurocomputing **461**, 370–403 (2021)
13. Chen, B., et al.: Physical mechanisms of endurance degradation in TMO-RRAM. In: International Electron Devices (2011)
14. Le, B.Q., et al.: Resistive RAM with multiple bits per cell: array-level demonstration of 3 bits per cell. IEEE Trans. Electron Dev. **66**(1), 641–646 (2018)
15. Yong, H., Huang, J., Hua, X., Zhang, L.: Gradient centralization: a new optimization technique for deep neural networks. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12346, pp. 635–652. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58452-8_37
16. Huang, Z., Shao, W., Wang, X., Lin, L., Luo, P.: Rethinking the pruning criteria for convolutional neural network. Adv. Neural Inf. Process. Syst. **34**, 16305–16318 (2021)
17. Molchanov, P., et al.: Pruning convolutional neural networks for resource efficient inference. In: International Conference on Learning Representations (2017)
18. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. In: Conference on Neural Information Processing Systems (1989)
19. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv Machine Learning (2017)
20. Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Comput. Sci. Eng. **13**(2), 22–30 (2011)
21. Zhang, S., et al.: Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects. In: IEEE International Conference on Artificial Intelligence Circuits and Systems (2020)
22. Tanaka, H., et al.: Pruning neural networks without any data by iteratively conserving synaptic flow. Adv. Neural Inf. Process. Syst. **33**, 6377–6389 (2020)