SCIENCE CHINA
Information Sciences

June 2021, Vol. 64 160407:1–160407:11
https://doi.org/10.1007/s11432-020-3245-7

• RESEARCH PAPER •
Special Focus on Near-memory and In-memory Computing

NAS4RRAM: neural network architecture search for inference on RRAM-based accelerators

Zhihang YUAN[1], Jingze LIU[1], Xingchen LI[1], Longhao YAN[2,3], Haoxiang CHEN[1], Bingzhe WU[1], Yuchao YANG[2,3] & Guangyu SUN[1*]

[1]*Center for Energy-Efficient Computing and Applications, Peking University, Beijing* 100871*, China;*
[2]*Department of Micro/nanoelectronics, Peking University, Beijing* 100871*, China;*
[3]*Center for Brain Inspired Chips, Institute for Artificial Intelligence, Peking University, Beijing* 100871*, China*

Received 31 December 2020/Revised 3 March 2021/Accepted 7 April 2021/Published online 10 May 2021

Abstract The RRAM-based accelerators enable fast and energy-efficient inference for neural networks. However, there are some requirements to deploy neural networks on RRAM-based accelerators, which are not considered in existing neural networks. (1) Because the noise problem and analog-digital converters/digital-analog converters (ADC/DAC) affect the prediction accuracy, they should be modeled in networks. (2) Because the weights are mapped to the RRAM cells, they should be quantized, and the number of weights is limited by the number of RRAM cells in the accelerator. These requirements motivate us to customize the hardware-friendly network for the RRAM-based accelerator. We take the idea of network architecture search (NAS) to design networks with high prediction accuracy that meet the requirements. We propose a framework called NAS4RRAM to search for the optimal network on the given RRAM-based accelerator. The experiments demonstrate that NAS4RRAM can apply to different RRAM-based accelerators with different scales. The performance of searched networks outperforms the manually designed ResNet.

Keywords network architecture search (NAS), neural networks, RRAM-based accelerator, hardware noise, quantization

Citation Yuan Z H, Liu J Z, Li X C, et al. NAS4RRAM: neural network architecture search for inference on RRAM-based accelerators. Sci China Inf Sci, 2021, 64(6): 160407, https://doi.org/10.1007/s11432-020-3245-7

1 Introduction

In the past years, neural networks have gained great success in a lot of tasks such as image classification [1] and object detection [2]. However, neural networks' performance always comes with a substantial computational cost, which impedes their deployment on real-world applications. Thus, various hardware accelerators have been proposed to support the parallel multiply-accumulate (MAC) calculations of network inference, such as GPGPU [3], FPGA [4] and ASIC [5]. To further improve computing performance and energy efficiency, the processing-in-memory (PIM) architecture based on RRAM technology is proposed to perform the parallel MACs in neural networks [6,7], which is called RRAM-based accelerator.

An RRAM-based accelerator is composed of RRAM crossbars. We distributed the weights of neural networks to the RRAM cells in these crossbars. In the RRAM crossbar, the digital-analog converters (DAC) transform the digital input data (e.g., the feature maps in convolutional neural networks) to analog voltages. The analog MACs are executed in the crossbar. The output currents represent the results of MACs, and the analog-digital converters (ADC) transform the currents into digital values. Since the MAC is directly accomplished by Ohm's law and Kirchhoff's law, analog computing in the crossbar is more efficient than digital computing to support the parallel MACs of neural networks.

However, there are some requirements for the neural networks to deploy on a given RRAM-based accelerator. (1) The weight in the neural network should be quantized since the number of RRAM cells' conductance levels is limited. (2) For a specific accelerator, the total number of RRAM cells is limited, constraining the number of weights. (3) Since DAC and ADC are used in the RRAM crossbar, and

* Corresponding author (email: gsun@pku.edu.cn)

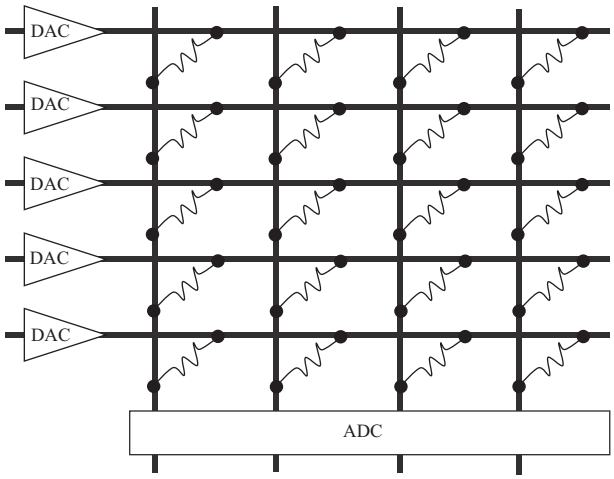

Figure 1 Demonstration of the RRAM crossbar.

the conversions affect the prediction accuracy, they should be modeled in the neural network. (4) The neural network should be noise resistant because the RRAM-based accelerator's inevitable noise affects the prediction accuracy. These requirements impede the deployment of the existing neural networks.

Since the existing neural networks can hardly meet these requirements, it is desirable to design hardware-friendly networks, which can be deployed on the given RRAM-based accelerator and perform good prediction accuracy under noise. However, manually designing such a network is a non-trivial task.

Recently, network architecture search (NAS) has emerged to automatically design the neural network [8]. This motivates us to use the powerful NAS to design the hardware-friendly network for the RRAM-based accelerator. We formulate this problem of network design and propose a framework, NAS4RRAM, to search for the optimal network given the RRAM-based accelerator. NAS4RRAM considers these requirements to build the search space in which each network can be deployed. The evolution algorithm is used to explore the search space, which simulates biological evolution, including reproduction, mutation, and selection. To compare different networks, we design a criterion that combines the prediction accuracy and the computation cost, and we propose a method to evaluate the sampled networks with hardware noise and DAC/ADC considered.

To verify the effectiveness of NAS4RRAM, we design a search space and search for the optimal network for three RRAM-based accelerators with different scales as an example. The results demonstrate that NAS4RRAM works well for different accelerators and different tasks. The prediction accuracy of the searched networks significantly outperforms the manually designed ResNet networks. Deploying the searched network, the utilization of the RRAM cells is high, which indicates NAS4RRAM can design hardware-friendly networks that make full use of the hardware resources.

2 Background and related work

A lot of RRAM-based accelerators have been proposed to support the inference of neural networks [6, 7, 9–16]. In this section, we first introduce the mechanism and the noise of the RRAM-based accelerator. Then we introduce the background of NAS and some related work.

2.1 Mechanism of accelerator

The basic organization of the RRAM-based accelerator is the RRAM crossbar, which can perform the matrix-vector product efficiently. The matrix-vector product is defined as

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}, \tag{1}$$

where $\boldsymbol{W}$ is an $n \times m$ matrix, $\boldsymbol{x}$ is a vector with m elements and $\boldsymbol{y}$ is a vector with n elements.

As demonstrated in Figure 1, an RRAM crossbar contains #Row×#Col RRAM cells and the peripheral devices. The weights ($\boldsymbol{W}$) in the matrix are mapped to the conductance of the RRAM cells $\boldsymbol{G}$. Since the computation is in analog, the input vector ($\boldsymbol{x}$) is transferred to analog voltage $\boldsymbol{V}$ using DAC. The analog

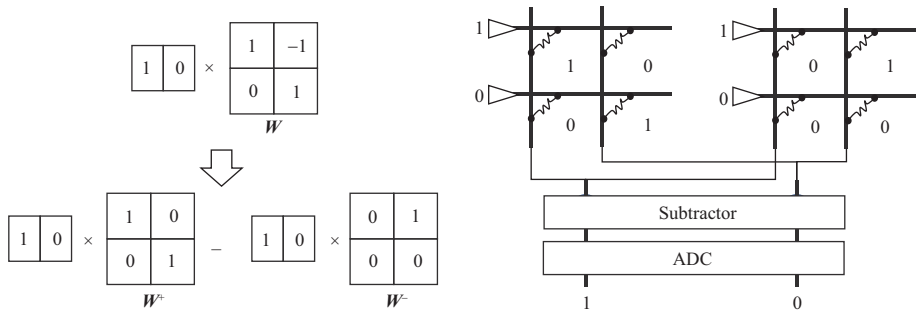

Figure 2 Demonstration of computation of matrix-vector product for the weight with negative value on RRAM crossbar.

computation is performed and the current flow to the end of the i-th column is $\boldsymbol{I}_i = \sum_j \boldsymbol{G}_{i,j}\boldsymbol{V}_j$. Then the current flow is converted to digital signals using ADC as the result of the matrix-vector product. Since the multiplication and accumulation are performed in analog, the RRAM crossbar is highly efficient to perform the parallel computation for a matrix-vector product.

Note that the conductance of RRAM cells should be non-negative. In order to enable the matrix-vector product for the matrix with both positive and negative numbers, we transform it into positive part $\boldsymbol{W}^+$, which takes the positive numbers, and negative part $\boldsymbol{W}^-$, which takes the absolute values of negative numbers. As demonstrated in Figure 2, we map them to the conductance of RRAM crossbar as $\boldsymbol{G}^+_{i,j}$ and $\boldsymbol{G}^-_{i,j}$, respectively. Therefore, we can perform the matrix-vector product for the two parts. The result of the positive part is subtracted by the result of the negative part, which is formulated as

$$\boldsymbol{I}_i = \sum_j \boldsymbol{G}^+_{i,j}\boldsymbol{V}_j - \sum_j \boldsymbol{G}^-_{i,j}\boldsymbol{V}_j. \tag{2}$$

2.2 Map network to accelerator

In this paper, we focus on the convolutional neural network, which is the most widely used network on the RRAM-based accelerator. The computation of convolution layers can be transformed into matrix-vector products in different sliding windows [9]. The weights of one output channel are divided into the positive part and the negative part. They are mapped to the conductance of RRAM cells of different columns in the crossbar. When the size of the matrix is larger than the size of the crossbar, we partition them into multiple crossbars. The weights of different output channels are mapped to different columns and they can compute simultaneously, which results in high throughput. Since the computational cost of other layers is not significant, they are processed digitally.

2.3 Noise sources

In the RRAM crossbar, the computations happen in the analog domain. Some inevitable noise sources affect the inference result of neural networks. The components related to the analog domain include DAC, RRAM cell, and ADC. In this subsection, we introduce the noise sources from these components respectively.

2.3.1 *DAC*

In most RRAM-based accelerator designs, a k_{in}-bits input activation is split to k_{in} 1-bit values, and processed by the crossbar in k_{in} cycles. Then, the partial results are put together by a shift-add circuit in the digital domain. Therefore, we take 1-bit DAC in our experiments. Because 1-bit DAC can be implemented as an on/off switch, its noise is insignificant and we ignore it in our noise model.

2.3.2 *RRAM cell*

The significant noise sources of the RRAM cell include thermal noise, shot noise, and random telegraph noise (RTN). Previous studies modeling the RRAM noise mainly focus on them [17,18]. Following their studies, we also consider these noise sources in our model. The thermal noise and shot noise are modeled as a Gaussian distribution $\mathcal{N}(0, \frac{Gf(4K_{\text{B}}T+2qV_{\text{drop}})}{V^2_{\text{drop}}})$, where G is the conductance of an RRAM cell, f is

the frequency of the crossbar, K_{B} is Boltzmann constant, T is temperature, q is the electron charge and V_{drop} is the drop of voltage. The RTN noise is modeled as a Poisson distribution $\{\begin{smallmatrix}\frac{bG+a}{G-(bG+a)}G & p{=}0.5\\ 0 & p{=}0.5\end{smallmatrix}$, where $a = 1.662\mathrm{E}-7$ and $b = 0.0015$ [19].

2.3.3 *ADC*

The noise of the ADC has been considered when deciding the ADC resolution. Knowing the signal-to-noise and distortion ratio (SINADR) of an ADC, the number of effective bits it can sample is $\frac{\mathrm{SINADR}}{6.02}$. Therefore, we also do not consider the noise of ADC in our noise model.

2.4 Network architecture search

In recent years, NAS [8, 20, 21] has emerged to automatically design the networks. Given the specific constraint, the algorithm searches for the optimal network in a predefined search space of various networks. The NAS is modeled as a sequence generation problem in [8], which can be solved by reinforcement learning. Besides the reinforcement learning-based algorithms, the evolutionary algorithm is also widely used in NAS [22, 23]. NAS4RRAM takes the evolutionary algorithm.

Since the computational efficiency of the network varies on different hardware devices, the hardware-aware NAS is proposed to search for the optimal network on a given hardware device [24–26]. Noise sources of hardware devices have been considered in some studies. FTT-NAS [27] models the bit flip noise on edge devices and searches for the fault-tolerant neural architecture. FTR-NAS [28] searches for the fault-tolerant recurrent neural architectures with weight faults and bit-flip noise. NACIM [29] proposed to use NAS to co-optimize the network and hardware including the processing-in-memory devices. Different from them, NAS4RRAM models the RRAM noise in detail and considers more hardware constraints (i.e., the number of weights and the ADC/DAC quantization).

3 Motivation

Different from other types of accelerators, there are some requirements for the network to deploy on the RRAM-based accelerator.

• **Weight quantization.** We can only write an RRAM cell to a small number of conductance states. So the weights of the network should be quantized.

• **Number of weights.** Given an RRAM-based accelerator, the number of available RRAM cells is fixed. Assuming the size of the crossbars is #Row × #Col, the number of RRAM cells is #Cells = $B \times$ #Row × #Col, where B is the number of the crossbars in the accelerator. Since both the positive part and negative part of the weights in a network should be mapped to the RRAM cells, the number of weights #Weight is limited, which is formulated as

$$\#\mathrm{Weight} \leqslant \frac{\#\mathrm{Cells}}{2} = \frac{B \times \#\mathrm{Row} \times \#\mathrm{Col}}{2}. \tag{3}$$

• **DAC/ADC.** The inputs of the crossbar are transformed from digital numbers to sequences of analog voltages by DAC and the length of the sequence is an integer k_{in}. ADC transforms the continuous current flow into the quantized digital value with a fixed number of bits k_{out}. Since the two conversions affect the result, they should be considered in the network.

• **Noise resistance.** The inevitable noise of the RRAM-based accelerator affects the prediction accuracy of the network. This requires the network can resist noise.

Existing networks are not designed with these requirements. For example, the total number of weights #Weight is not limited. As a result, these networks cannot be well deployed on the RRAM-based accelerator. Networks are not deployable when the #Weight is out of the limit. The other case is that they cannot make full use of hardware resources when the #Weight is too small because many RRAM cells are idle. Furthermore, the prediction accuracy is not guaranteed because the inevitable noise, weight quantization, and DAC/ADC are not considered in the existing networks.

To fully utilize the hardware resources and guarantee prediction accuracy, we should design hardware-friendly networks considering these requirements. However, manually designing a network is a non-trivial

task. This motivates us to take the powerful NAS to design the networks for deploying on a given RRAM-based accelerator. Therefore, we propose the framework, NAS4RRAM, which is used to search for the optimal network to meet these requirements.

4 Problem formulation

Given an RRAM-based accelerator, the goal is to find a deployable network $\boldsymbol{a}$ with high prediction accuracy and low computation costs. We first formulate this problem by defining the deployable network, introducing the criterion, and formulate the search.

To deploy a network on the RRAM-based accelerator, we should map the weights to the RRAM cells' conductance in different crossbars. Therefore, the weights should be quantized, and the total number of the weights should not be greater than the number of the RRAM cells #Cells in the accelerator. We define a network $\boldsymbol{a}$ is deployable under the following constraints:

$$W_{\boldsymbol{a}}^l \in \mathcal{Q}^l = \{q_1^l, q_2^l, \ldots, q_n^l\}, \tag{4}$$

$$\sum_l |W_{\boldsymbol{a}}^l| \leqslant \frac{\#\text{Cells}}{2}, \tag{5}$$

where $W_{\boldsymbol{a}}^l$ is the weights in the l-th convolution layer of the network $\boldsymbol{a}$, $\mathcal{Q}^l = \{q_1^l, q_2^l, \ldots, q_n^l\}$ is the set of the quantization points, where n is the number of conductance levels of RRAM-cells, and the $|W_{\boldsymbol{a}}^l|$ is the number of elements in $W_{\boldsymbol{a}}^l$.

Next, we introduce the criterion to evaluate different networks. The criterion takes the prediction accuracy and computation cost of the network into consideration. Since the prediction accuracy of the network is affected by the inevitable noise ϵ, we take the expected accuracy under the noise $E_\epsilon(\text{ACC}_{\boldsymbol{a}})$. The computation cost $\text{COST}_{\boldsymbol{a}}$ of the network can be the energy consumption or latency of the network inference. Boosting prediction accuracy often accompanies the increase in computation cost. Therefore, we take a criterion that combines the prediction accuracy and computation cost:

$$S_{\boldsymbol{a}} = \frac{E_\epsilon(\text{ACC}_{\boldsymbol{a}})}{\text{COST}_{\boldsymbol{a}}^\omega}, \tag{6}$$

where $S_{\boldsymbol{a}}$ is the score of the network $\boldsymbol{a}$ and ω is the parameter that controls the trade-off between the prediction accuracy and the computation cost.

Using this criterion, we can search for the optimal deployable network with the highest score S^*. However, the number of available networks is infinite, which cannot be well handled by any search algorithm. Therefore, we limit the search space to a finite set $\mathcal{A}$. The problem of network search in the search space $\mathcal{A}$ is formulated as

$$\begin{aligned} &S^* = \operatorname*{argmax}_{\boldsymbol{a}} S_{\boldsymbol{a}} \\ &\text{s.t. } \boldsymbol{a} \in \mathcal{A},\ W_{\boldsymbol{a}}^l \in \mathcal{Q},\ \sum_l |W_{\boldsymbol{a}}^l| \leqslant \frac{\#\text{Cells}}{2}. \end{aligned} \tag{7}$$

In this way, we incorporate the requirements of the network to be deployed on the RRAM-based accelerators into the formulation.

5 NAS4RRAM framework

In this section, we introduce the framework, NAS4RRAM. The overall is demonstrated in Figure 3. The search algorithm samples networks from the search space. Then we evaluate the sampled networks using the evaluation metric to get the score of each network. And we feed the scores back to the search algorithm to sample better networks. After some iterations of sample/evaluation/feedback, we take the best network explored by the search algorithm. We can deploy it to the RRAM-based accelerator.

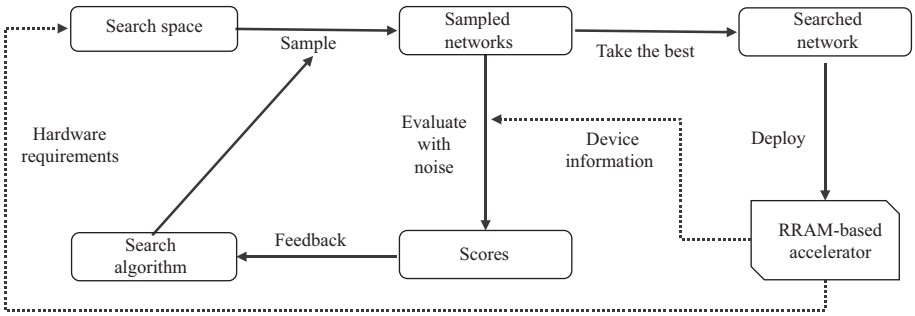

Figure 3 Overview of the NAS4RRAM framework.

5.1 Search space

The search space is a finite set that contains a lot of deployable networks. The goal of NAS is to find the optimal network in the search space, and the networks not in the space are ignored. Since the search space determines the range of network search, it is essential to build a proper search space $\mathcal{A}$.

There are many available choices to build the search space, including (1) the number of layers, (2) the topology of layers, (3) the parameters of layers (e.g., the number of channels, the kernel size, and the group size for convolution layer). NAS4RRAM requires the search space to contain networks that are deployable on the given RRAM-based accelerator. For example, a network with an excessive number of weights is forbidden to be in the search space.

A large space contains many networks, in which the probability of finding a powerful network is increased. However, search algorithms cannot well explore a large space given finite computation resources [30]. A small space contains fewer networks, in which the probability of finding a powerful network is decreased. So we should use a search space with a proper size. We design a search space with around 10^{16} networks in our experiment as an example described in Subsection 6.1.2.

5.2 Search algorithm

The search algorithm's goal is to explore the search space $\mathcal{A}$ and search for the optimal network with the highest score S^*. There exist different kinds of search algorithms, including reinforcement-learning based algorithm [8], evolutionary algorithm [23] and Bayesian optimization [31]. In NAS4RRAM, we take the widely used evolutionary algorithm, which is powerful and easy to implement. The algorithm simulates biological evolution, which includes the steps of reproduction, mutation, and selection. Next, we introduce the search algorithm in detail.

In the evolutionary algorithm, a population is some networks sampled from the search space. The network in the population is also called individual. The population can evolve iteratively. Like natural selection, a number of good networks in the population are selected as the parent. The other networks cannot survive, and they are deleted from the population. By mutation in the network structures of the parent networks, the child networks can be generated. The generated child networks update the population. Therefore, the networks are evolved iteratively. At natural selection, we take the proposed criterion to evaluate each network in the population, and the scores of the networks can increase during the evolution.

Specifically, we first random sample #Individual networks to initialize the population. Then we repeat the following steps #Evolution times.

(1) Evaluate the networks in the population to get the scores.

(2) Select #Parent networks with the highest scores as parent networks and delete the other networks.

(3) Generate (#Individual − #Parent) child networks by mutating the parent networks.

(4) Put the generated networks into the population.

At last, we take the network with the highest score as output. The mutation operation is implemented by randomly changing the parent network configurations to generate a new network in the search space. For example, the number of output channels of a convolution layer in a parent network can be changed to generate another network.

5.3 Evaluation of networks

When the child networks are sampled from the search space, we should evaluate them to get the score. The score $S_{\boldsymbol{a}}$ of network $\boldsymbol{a}$ is calculated by the expected prediction accuracy $E_\epsilon(\mathrm{ACC}_{\boldsymbol{a}})$ and the computation cost $\mathrm{COST}_{\boldsymbol{a}}$ according to the criterion in (6).

Generally, NAS takes the following steps to get the prediction accuracy.

(1) Train the network using a training dataset (training phase).

(2) Test the trained network using an evaluation dataset and calculate the prediction accuracy (testing phase).

Different from digital hardware, the noise in the RRAM-based accelerator cannot be ignored, and the DAC and ADC are used to transform the input and output of the crossbar between digital numbers and analog signals. The noise and the DAC/ADC affect the prediction accuracy, which should be considered here.

We model the noise in both the training phase and testing phase. The noise of the RRAM-based accelerator is described in Subsection 2.3. The thermal noise, the shot noise, and the RTN noise are added to the convolution layers' weight at each training iteration. At the testing phase, we test on the evaluation dataset with different randomly sampled noise multiple times. The prediction accuracy of different tests is averaged as an approximation of the expected prediction accuracy $E_\epsilon(\mathrm{ACC}_{\boldsymbol{a}})$.

The same as [32], we model the DAC and the ADC in the RRAM crossbar as two quantization functions, respectively. The DAC is modeled as a k_{in} bits quantization function, and the ADC is modeled as a k_{out} bits quantization function. At the training phase, the training techniques for quantization can be used such as straight-forward estimator [33] or error decay estimator [34]. Note that the weights in each layer can be mapped to several crossbars, and the quantization functions are applied at each crossbar, which is different from the layer-wise or channel-wise quantization.

The computation cost of the network is calculated using a simulator of the target RRAM-based accelerator. We first collect the latency and energy consumption of matrix-vector multiplication on different crossbars. The sampled networks are mapped to the crossbars and calculate the total latency and energy consumption.

6 Experiments

To verify the effectiveness of NAS4RRAM, we search for the optimal networks on the CIFAR-10 and the CIFAR-100 classification tasks to deploy on the RRAM-based accelerators with different sizes. In this section, we first describe the settings, including the accelerator's details, the design of the search space, the hyper-parameters of NAS, and the dataset. Then we demonstrate the results of the experiments.

6.1 Settings

6.1.1 *RRAM-based accelerator*

In our experiments, the RRAM cells in the accelerator can be programmed to 2 levels of conductance. This 2-level RRAM cell can achieve high energy efficiency and low noise, which is widely used. Therefore, we use the ternary weight, of which the value is selected from $\{-1, 0, 1\}$. Each weight consumes two RRAM cells because we divide the weights in the matrix into the positive and negative parts. To enable quick inference and low energy consumption, we chose $k_{\mathrm{in}} = 1$ and $k_{\mathrm{out}} = 4$. The operating frequency f is 100 MHz, the voltage drop V_{drop} is 0.2 V, the 2 levels of conductance are 333 and 0.33 μS, the temperature T is 300 K, and the crossbar size #Row × #Col is 128 × 128. We set different numbers of the crossbars B to scale the RRAM-based accelerator. In our experiment, we take $B \in \{16, 32, 48\}$. Therefore, the numbers of weights in the networks are limited to 131072, 262144, and 393216, respectively.

6.1.2 *Search space*

In our experiments, we design a search space based on ResNet [1]. A ResNet is composed of a number of residual blocks, demonstrated in Figure 4(a). It contains a branch with two convolution layers and a residual connection. Inspired by IRNet [34], we modify the standard residual block to support the low bit-width quantization. As shown in Figure 4(b), we remove one of the convolution layers, change the number of channels, and replace the activation function with Hardtanh, which is defined as $\mathrm{Hardtanh}(x) =$

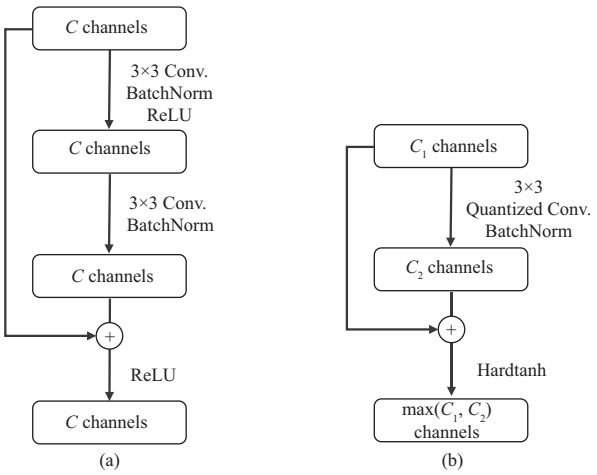

Figure 4 Comparing of (a) standard residual block and (b) the modified residual block.

$\min(\max(x, -1), 1)$. The input and output numbers of channels are C_1 and C_2, respectively. At the residual connection, we add the feature maps with fewer channels to the front channels of the other one if $C_1 \neq C_2$. If the convolution's stride is 2, we down-sample the input feature maps to align the resolution. Note that the weights of the 3×3 convolution are quantized, and the ADC/DAC quantization functions are used.

As shown in Figure 5, we build the search space by stacking the modified residual blocks. The modified residual blocks are divided into three block groups. Block_i^g is the i-th residual block in g-th block group. C_i^g is the number of the block's output channel. N_g is the number of blocks in the block group g. The dimensions of the search space are (1) the numbers of layers $N_g \in \mathbb{N}$ and (2) the numbers of output channels $C_i^g \in \mathbb{C}$ for different convolutions. We set $\mathbb{C} = \{16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64\}$ and $\mathbb{N} = \{2, 4, 6, 8, 10\}$. We set the first block in the second and the third group with stride $= 2$ to reduce the resolution of the feature maps. We add a convolution layer with 16 output channels before the first residual block, and we add a global average pooling followed by a fully connected layer after the last residual block. Since the quantization for the first convolution layer and the fully connected layer dramatically decrease the model performance, they are processed in the CPU with floating-point values.

6.1.3 *Hyper-parameters of NAS*

For the evolution algorithm, we set the population size #Individual $= 200$, the parent size #Parent $= 50$ and the time of evolution #Evolution $= 20$. Therefore, $200+20\times(200-50) = 3200$ networks in the search space are sampled during the evolutionary search. We set the mutation probability of each configuration to 20%.

At the training phase of evaluation, we use SGD as the optimizer to train the sampled networks. The momentum is set to 0.9. The batch size is set to 256. The learning rate is set to 0.1. The weight decay is set to $1\text{E}-4$. Each sampled network is trained for 10 epochs. At the testing phase, we test it under the affection of noise 5 times to calculate the expected prediction accuracy. We use a GPU server with 6 Nvidia Tesla V100 to train these networks. We use the energy consumption as the computation cost, estimated by a simulator of the RRAM-based accelerator. The ω in (6) is set to 0.06.

After the network search, we train the searched network for 200 epochs. The learning rate starts from 0.1, and cosine annealing schedule [35] is used. The error decay estimator [34] is used to train with quantization.

6.1.4 *Dataset*

Both CIFAR-10 and CIFAR-100 [36] contain 50k training images and 10k test images. We sample 5000 images from the training dataset as the evaluation dataset. The input resolution of the images is 32×32. The standard data augmentation is used. We zero-pad the image with 4 pixels, randomly crop it to the resolution of 32×32, and randomly horizontal flip it as data augmentation.

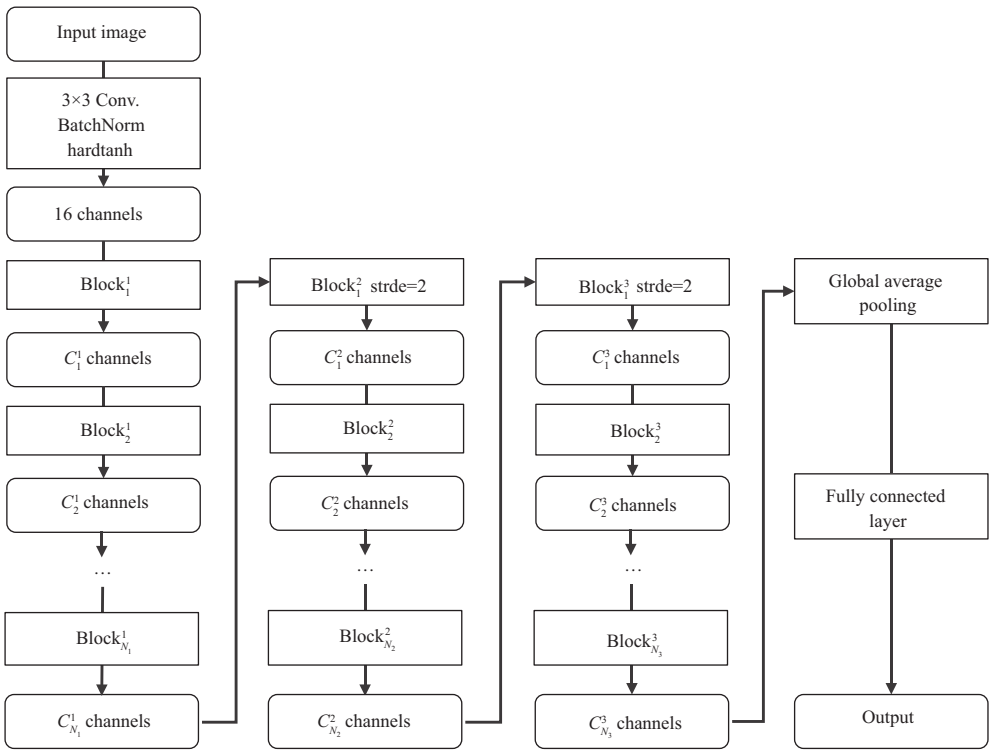

Figure 5 Demonstration of the networks in the search space.

Table 1 The results on CIFAR-10/CIFAR-100 for different networks[a)]

Task	Network	#Weight	ACC (%)	Deployable ($B = 16$)	Deployable ($B = 32$)	Deployable ($B = 48$)
CIFAR-10	ResNet-20 ×1	267k	82.4	N	N	Y
	ResNet-20 ×0.5	71k	72.6	Y	Y	Y
	ResNet-32 ×1	461k	82.9	N	N	N
	ResNet-32 ×0.5	122k	76.1	Y	Y	Y
	NAS4RRAM (B =16)	125k	78.5	Y	Y	Y
	NAS4RRAM (B =32)	261k	82.7	N	Y	Y
	NAS4RRAM (B =48)	383k	84.4	N	N	Y
CIFAR-100	ResNet-20 ×1	267k	50.7	N	N	Y
	ResNet-20 ×0.5	71k	38.2	Y	Y	Y
	ResNet-32 ×1	461k	53.0	N	N	N
	ResNet-32 ×0.5	122k	39.3	Y	Y	Y
	NAS4RRAM ($B = 16$)	118k	45.6	Y	Y	Y
	NAS4RRAM ($B = 32$)	250k	50.9	N	Y	Y
	NAS4RRAM ($B = 48$)	343k	53.1	N	N	Y

a) #Weight is the number of weights in thousand. ACC is the top-1 accuracy. We mark a Y at the corresponding column if the network is deployable on an accelerator with B RRAM-crossbars.

6.2 Results

We search for the optimal networks to deploy on accelerators with different numbers of RRAM crossbars B. The searched networks are denoted as NAS4RRAM ($B = 16$), NAS4RRAM ($B = 32$), and NAS4RRAM ($B = 48$) for the accelerators with 16, 32, and 48 RRAM crossbars, respectively.

We compare the searched networks with the manually designed ResNet, denoted by ResNet-20 ×1 and ResNet-32 ×1. However, they are too large to deploy on the RRAM-based accelerator with $B = 16$ and $B = 32$. We halve their output channel of each convolution layer to generate smaller networks, denoted as ResNet-20 ×0.5 and ResNet-32 ×0.5. They are evaluated with the same settings (i.e., noise affection, quantization, training settings, and testing settings).

As shown in Table 1, we report the number of weights and the prediction accuracy of different networks. The ResNet-20 ×1 takes 267k weights and is only deployable on the largest accelerator ($B = 48$). And

ResNet-20 ×0.5 is much smaller than ResNet-20 ×1. It takes 71k weights and is deployable on the three accelerators. However, the utilization of the RRAM cells is low. The utilization is 54.2% when the ResNet-20 ×0.5 is deployed on the smallest accelerator ($B = 16$). The ResNet-32 ×1 is a large network that takes 461k weights. We cannot deploy it on any accelerators. The ResNet-32 ×0.5 takes 122k weights and can be deployed on the three accelerators.

The results demonstrated the networks searched by NAS4RRAM are deployable on the target accelerators. On CIFAR-10, the NAS4RRAM ($B = 16$) takes 122k weights, and its prediction accuracy is 78.5%. Comparing with ResNet-20 ×0.5 (72.6%) and ResNet-32 ×0.5 (76.1%), the prediction accuracy of NAS4RRAM ($B = 16$) is much higher. The NAS4RRAM ($B = 32$) takes 261k weights with the prediction accuracy of 82.7%, which outperforms the other networks that can be deployed on the accelerator ($B = 32$). Targeting the accelerator with 48 RRAM crossbars, the NAS4RRAM ($B = 48$) takes 383k weights with the highest prediction accuracy (84.4%).

On CIFAR-100, the NAS4RRAM ($B = 16$) takes 118k weights, and its prediction accuracy is 45.6%. Comparing with ResNet-20 ×0.5 (38.2%) and ResNet-32 ×0.5 (39.3%), which are deployable on accelerator ($B = 16$), the prediction accuracy of NAS4RRAM ($B = 16$) is much higher. The NAS4RRAM ($B = 32$) takes 250k weights with a prediction accuracy of 50.9%. Targeting the accelerator with 48 RRAM crossbars, the NAS4RRAM ($B = 48$) takes 343k weights with the highest prediction accuracy (53.1%).

We observe that the RRAM cells' utilization is much higher for the searched networks, which indicates NAS4RRAM can make full use of the hardware devices.

7 Conclusion

In this paper, we observed the requirements of deploying a neural network on RRAM-based accelerators. The requirements motivate us to design hardware-friendly networks. Taking the technique of NAS, we propose a framework, NAS4RRAM. It searches for the optimal network to deploy on the given RRAM-based accelerator. As an example, we validate the effectiveness of NAS4RRAM on the CIFAR-10 and CIFAR-100 classification tasks. The experimental results demonstrate that the NAS4RRAM can be applied on different RRAM-based accelerators with different scales. The searched networks outperform the manually designed ResNet.

Acknowledgements This work was supported by National Key Research and Development Project of China (Grant No. 2018YFB-1003304) and National Natural Science Foundation of China (Grant Nos. 61832020, 62032001).

References

1 He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, 2016. 770–778
2 Ren S, He K, Girshick R B, et al. Faster R-CNN: towards real-time object detection with region proposal networks. In: Proceedings of Annual Conference on Neural Information Processing Systems 2015, Montreal, 2015. 91–99
3 Coates A, Huval B, Wang T, et al. Deep learning with COTS HPC systems. In: Proceedings of the 30th International Conference on Machine Learning, Atlanta, 2013. 1337–1345
4 Zhang C, Li P, Sun G, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2015. 161–170
5 Chen Y, Luo T, Liu S, et al. Dadiannao: a machine-learning supercomputer. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, 2014. 609–622
6 Shafiee A, Nag A, Muralimanohar N, et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. SIGARCH Comput Archit News, 2016, 44: 14–26
7 Chi P, Li S, Xu C, et al. PRIME: a novel processing-in- memory architecture for neural network computation in ReRAM-based main memory. SIGARCH Comput Archit News, 2016, 44: 27–39
8 Zoph B, Le Q V. Neural architecture search with reinforcement learning. In: Proceedings of the 5th International Conference on Learning Representations, Toulon, 2017
9 Song L, Qian X, Li H, et al. Pipelayer: a pipelined ReRAM-based accelerator for deep learning. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017. 541–552
10 Ji Y, Zhang Y, Xie X, et al. FPSA: a full system stack solution for reconfigurable ReRAM-based NN accelerator architecture. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems, 2019. 733–747
11 Tang S, Yin S, Zheng S, et al. AEPE: an area and power efficient RRAM crossbar-based accelerator for deep CNNs. In: Proceedings of IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA), 2017. 1–6
12 Liu X, Mao M, Liu B, et al. RENO: a high-efficient reconfigurable neuromorphic computing accelerator design. In: Proceedings of the 52nd Annual Design Automation Conference, 2015. 1–6
13 Zhu Z, Sun H, Lin Y, et al. A configurable multi-precision CNN computing framework based on single bit RRAM. In: Proceedings of 56th ACM/IEEE Design Automation Conference (DAC), 2019. 1–6
14 Zhu Z, Lin J, Cheng M, et al. Mixed size crossbar based RRAM CNN accelerator with overlapped mapping method. In: Proceedings of 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018. 1–8

15 Umesh S, Mittal S. A survey of spintronic architectures for processing-in-memory and neural networks. J Syst Architecture, 2019, 97: 349–372
16 Mohamed K S. Near-memory/in-memory computing: pillars and ladders. In: Proceedings of Neuromorphic Computing and Beyond, 2020. 167–186
17 He Z, Lin J, Ewetz R, et al. Noise injection adaption: end-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping. In: Proceedings of the 56th Annual Design Automation Conference, Las Vegas, 2019. 57
18 Feinberg B, Wang S, Ipek E. Making memristive neural network accelerators reliable. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture, Vienna, 2018. 52–65
19 Puglisi F M, Larcher L, Padovani A, et al. A complete statistical investigation of RTN in HfO_2-based RRAM in high resistive state. IEEE Trans Electron Devices, 2015, 62: 2606–2613
20 Zoph B, Vasudevan V, Shlens J, et al. Learning transferable architectures for scalable image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, 2018. 8697–8710
21 Zhong Z, Yan J, Wu W, et al. Practical block-wise neural network architecture generation. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, 2018. 2423–2432
22 Liu H, Simonyan K, Vinyals O, et al. Hierarchical representations for efficient architecture search. In: Proceedings of the 6th International Conference on Learning Representations, Vancouver, 2018
23 Real E, Aggarwal A, Huang Y, et al. Regularized evolution for image classifier architecture search. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence, the 31st Innovative Applications of Artificial Intelligence Conference, the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, Honolulu, 2019. 4780–4789
24 Tan M, Chen B, Pang R, et al. MnasNet: platform-aware neural architecture search for mobile. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, 2019. 2820–2828
25 Wang H, Wu Z, Liu Z, et al. HAT: hardware-aware transformers for efficient natural language processing. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020. 7675–7688
26 Wang T, Wang K, Cai H, et al. APQ: joint search for network architecture, pruning and quantization policy. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, 2020. 2075–2084
27 Li W, Ning X, Ge G, et al. FTT-NAS: discovering fault-tolerant neural architecture. In: Proceedings of the 25th Asia and South Pacific Design Automation Conference, Beijing, 2020. 211–216
28 Hu K, Ding D, Tian S, et al. FTR-NAS: fault-tolerant recurrent neural architecture search. In: Proceedings of the 27th International Conference on Neural Information Processing, Bangkok, 2020. 589–597
29 Jiang W, Lou Q, Yan Z, et al. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. 2019. ArXiv:1911.00139
30 Xie L, Chen X, Bi K, et al. Weight-sharing neural architecture search: a battle to shrink the optimization gap. 2020. ArXiv:2008.01475
31 Kandasamy K, Neiswanger W, Schneider J, et al. Neural architecture search with Bayesian optimisation and optimal transport. In: Proceedings of Annual Conference on Neural Information Processing Systems, Montréal, 2018. 2020–2029
32 Cai Y, Tang T, Xia L, et al. Low bit-width convolutional neural network on RRAM. IEEE Trans Comput-Aided Des Integr Circ Syst, 2020, 39: 1414–1427
33 Courbariaux M, Bengio Y, David J. Binaryconnect: training deep neural networks with binary weights during propagations. In: Proceedings of Annual Conference on Neural Information Processing Systems Montreal, 2015. 3123–3131
34 Qin H, Gong R, Liu X, et al. Forward and backward information retention for accurate binary neural networks. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, 2020. 2247–2256
35 Loshchilov I, Hutter F. SGDR: stochastic gradient descent with warm restarts. In: Proceedings of the 5th International Conference on Learning Representations, Toulon, 2017
36 Krizhevsky A. Learning Multiple Layers of Features From Tiny Images. Technical Report. Toronto: University of Toronto, 2009