

# CompRRAE: RRAM-based Convolutional Neural Network Accelerator with Reduced Computations through a Runtime Activation Estimation

Xizi Chen, Jingyang Zhu, Jingbo Jiang and Chi-Ying Tsui

Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong

**Abstract**– Recently Resistive-RAM (RRAM) crossbar has been used in the design of the accelerator of convolutional neural networks (CNNs) to solve the memory wall issue. However, the intensive multiply-accumulate computations (MACs) executed at the crossbars during the inference phase are still the bottleneck for the further improvement of energy efficiency and throughput. In this work, we explore several methods to reduce the computations for the RRAM-based CNN accelerators. First, the output sparsity resulting from the widely employed Rectified Linear Unit is exploited, and a significant portion of computations are bypassed through an early detection of the negative output activations. Second, an adaptive approximation is proposed to terminate the MAC early when the sum of the partial results of the remaining computations is considered to be within a certain range of the intermediate accumulated result and thus has an insignificant contribution to the inference. In order to determine these redundant computations, a novel runtime estimation on the maximum and minimum values of each output activation is developed and used during the MAC operation. Experimental results show that around 70% of the computations can be reduced during the inference with a negligible accuracy loss smaller than 0.2%. As a result, the energy efficiency and the throughput are improved by over 2.9 and 2.8 times, respectively, compared with the state-of-the-art RRAM-based accelerators.

## I. INTRODUCTION

Convolutional neural networks (CNNs) have demonstrated impressive performance in various machine learning tasks such as the visual recognition [11] and visual tracking [16]. At the same time, due to the nature of the convolutional operation, the inference of CNN usually involves intensive computations which are energy consuming and become a big deterrent for deploying CNN in embedded systems. Besides the high computation cost, conventional accelerators also face the memory wall issue where the massive memory accesses for fetching the weights and activations vastly limit the performance. Therefore, it is necessary to deliver a more efficient implementation with fewer computations.

The Rectified Linear Unit (ReLU) [5] has become the most widely used activation function in neural networks in recent years. Due to the application of ReLU, a high activation sparsity can be achieved during the inference [2]. Since the negative MAC results will be clamped to zero by ReLU, their actual magnitude values are irrelevant for the cascading layers. Thus, a large portion of computations corresponding to the negative output activations can be bypassed once the sign can be determined early. In addition, the inherent resilience of CNN makes the activation values error-tolerant to some degree, hence making it possible to reduce the computations by approximation without affecting the classification accuracy. To trigger the above computation bypass, we propose a runtime estimation on the maximum and minimum values of each output activation. During each MAC, the contribution of the intermediate accumulated result is evaluated continually against the estimated values of the remaining partial results. Once the contribution of the current accumulated result is considered

to be large enough to dictate the final result value, the MAC will be terminated to improve the energy efficiency and the throughput. The proposed methods are implemented in a specialized architecture based on the resistive random access memory (RRAM) crossbar [18] which utilizes the in-situ computation as an approach to address the high power density and the memory wall issue of the conventional CMOS-based design. In summary, the contributions of this work are as follows:

- A runtime estimation on the maximum and minimum values of the output activation is proposed and implemented during each MAC.
- By detecting the negative output activations through the estimation, the corresponding MACs are terminated in advance in the convolutional layers followed by ReLU. According to the experimental results, over 99.98% of negative outputs are detected and over 71.5% of their computations are bypassed without inducing accuracy loss.
- An adaptive approximation is proposed to bypass the remaining computations during the MAC when the estimated values of the remaining partial results are determined to have a negligible contribution to the inference.
- A dedicated RRAM-based architecture is proposed for implementing the CNNs with reduced computations. A total computation reduction of around 70% is achieved for the general 16-bit fixed-point implementation, and 40% reduction is achieved for the 8-bit implementation which demonstrates the effectiveness of the proposed methods under an aggressive quantization scheme. The induced accuracy loss is smaller than 0.2%. Experimental results show significant improvement in the energy efficiency and throughput.

## II. RELATED WORKS

Various techniques have been proposed to reduce the intensive computations in the CNN accelerators. A natural way for reducing the memory footprint and the number of multiplications in the CMOS-based accelerators is to utilize the activation sparsity [1, 2, 19]. Since a large portion of input activations are zero, the corresponding multiplications can be bypassed to save energy and time [2]. A recent work in [1] focuses on the layers with only non-negative inputs and exploits the output sparsity by reordering the weights to calculate the sum of the positive products first. Later calculation of the negative products will be terminated as soon as the accumulated result becomes smaller than zero. As a result, 16% energy saving and 28% speedup can be achieved. In a more aggressive mode, an empirical value is used to compare with the accumulated result after a specific number of multiplications. If the accumulated result is smaller, the output activation is considered likely to be negative. By bypassing the remaining multiplications, a higher reduction in complexity can be achieved, but a relatively large accuracy loss (3.0%) is induced. Another work in [19] exploits the output sparsity by using a low-rank approximation of the weight matrix to predict the output sparsity and disabling the actual computation if the predicted output is negative. In this case,

each MAC needs to be broadcast to all the processing elements to improve the throughput. Such methods, however, are not suitable for the RRAM-based architecture. Since the computations are executed at the RRAM crossbars where the weight matrix is programmed into the memristors before the classification, the multiplication-accumulation has to be done in a regular pattern. Therefore, it is difficult to irregularly skip the zero inputs, independently reorder the weights of each kernel, or broadcast different weight matrices to the crossbars at runtime. A way to reduce the computations in the RRAM-based architecture is to structurally compress the weights through training and then exploit the weight sparsity [7]. However, to the best of our knowledge, the output sparsity hasn't been efficiently exploited in the RRAM-based architecture. Due to the resilience of CNN, reducing the quantization bit-width of the weights and activations is another method for reducing computations [14]. A dynamic quantization scheme is proposed in [3] to change the bit-width of the weights when multiplying with the different bit of the activations to reduce the computations for the RRAM-based MAC.

### III. PRELIMINARIES

#### A. Convolutional Neural Networks

A convolutional neural network (CNN) [11] is a machine learning model inspired by the structure of the human brain. It is usually comprised of a series of cascading layers, including the convolutional (CONV) layers, pooling layers, and fully-connected (FC) layers. The CONV and FC layers consist of neurons to extract the features of the image. Inside each layer, the input activations from the previous layer are firstly multiplied with the corresponding weights in the current layer, and then accumulated to generate the output activations for the next layer. The computation of the CONV layer is shown in Fig.1(a) and can be expressed as follows:

$$a_{out}(x, y, z) = f\left(\sum_{l=0}^{c-1} \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} a_{in}(x+m, y+n, l) \times K_z(m, n, l)\right) \quad (1)$$

where  $a_{out}$  and  $a_{in}$  represent the output and the input activations, respectively;  $K_z$  represents the  $z^{th}$  kernel;  $h$ ,  $w$  and  $c$  represent the height, width, and depth of the kernel;  $(x, y, z)$ ,  $(x+m, y+n, l)$  and  $(m, n, l)$  represent the positions of the activations and weights in height, width, and depth.  $f$  is a non-linear activation function to avoid overfitting. The most commonly used activation function is ReLU given by:

$$f(x) = \max(0, x) \quad (2)$$

where  $x$  is the input to the function. FC layers are similar to the CONV layers, but have much fewer computations which can be simplified to a single vector-matrix multiplication. Pooling layers usually follow the CONV layers for down-sampling. In this work we will focus on the CONV layers since they account for most of the computations in CNN.

#### B. RRAM Crossbar and In-Situ Computation

The RRAM crossbar has aroused great research interest due to its high-density, non-volatility, and the potential for parallel in-situ analog computation [17, 18, 6]. While the CMOS-based accelerators face the difficulty of scaling down and the issue of memory wall, the RRAM-based computation provides a promising approach to achieve substantial improvement in the energy efficiency and throughput. For instance, the RRAM-based accelerator in [18] demonstrates 22 times energy saving compared with the CMOS-based counterpart. A hierarchical RRAM-based architecture proposed in [17] improves the

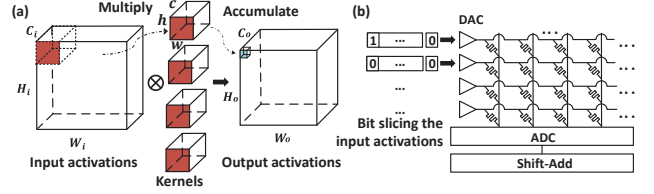


Fig. 1. (a) Computation of the CONV Layer; (b) In-Situ Computation based on the RRAM Crossbar.

energy efficiency and throughput by 5.5 and 14.8 times, respectively, compared with the state-of-the-art CMOS-based DaDianNao architecture [4]. The RRAM crossbar for the vector-matrix multiplication is shown in Fig.1(b). The elements of the weight matrix are stored as the conductance values of the memristors at the crosspoints connecting the horizontal wordlines and the vertical bitlines. When the computation starts, the input activation vector is applied on the wordlines as voltages. The current flowing through each memristor is equal to the product of the memristor conductance and the wordline voltage. Currents on the same bitline will be accumulated and output as the computation result. Since the computation is done in analog, digital-to-analog converters (DACs) are needed at the wordlines to convert the input activations to voltages, and analog-to-digital converters (ADCs) are needed at the bitlines to convert the results back to digital values. These interfacing circuits are the most energy consuming part during the computation [17, 3]. Since hundreds of products are accumulated vertically, the resolution requirement of ADC can easily go beyond the acceptable range and induce a huge energy overhead. As a common solution, the multi-bit multiplication, e.g. 16-bit multiplication, is broken into a series of low bit-width multiplications to limit the ADC resolution within a reasonable range [6, 17, 3]. For example, for a crossbar with 128 wordlines, the resolution of each crosspoint should be no more than 2-bit to keep the ADC resolution less than 10-bit. Thus, each weight takes multiple memristors to store. At the same time, since the multi-bit DAC is expensive to implement and hundreds of DAC operations are needed for one MAC, it is more efficient and preferable to use the single-bit DAC to minimize the overhead [17, 3]. Thus, each bit of the input activation is sent into the crossbar sequentially to finish the MAC. The whole MAC operation will take multiple iterations. At each iteration, a partial result corresponding to the current 1-bit input vector is generated, and then accumulated with the existing results of previous iterations. This bit-level slicing of activations creates a special scheduling for MAC and we will utilize this characteristic to reduce the computations.

### IV. RRAM-BASED COMPUTATION REDUCED ACCELERATOR DESIGN

#### A. Algorithms for Computation Reduction

In the CMOS-based accelerators, the MAC is usually done by accumulating the corresponding activation-weight products. However, since the activations are sliced into bit-level in the RRAM-based architecture, only a partial result is obtained at each iteration by accumulating the input bit-weight products. As the MAC sequentially proceeds from the most-significant bit (MSB) to the least-significant bit (LSB) of the input activations, the generated partial results also become less and less significant. Each new generated partial result will be accumulated with the sum of the partial results of previous iterations. An example is given in Fig.2 to illustrate this computation process, where the inner product of the activations [4, 12, 10] and the weights [4, -8, -5] is computed in 4 iterations. The intermediate accumulated result is updated at each iteration, represented as **Accu** [-104, -120, -130, -130] in Fig.2. Such bit-level processing makes it possible to terminate the MAC in advance once the re-

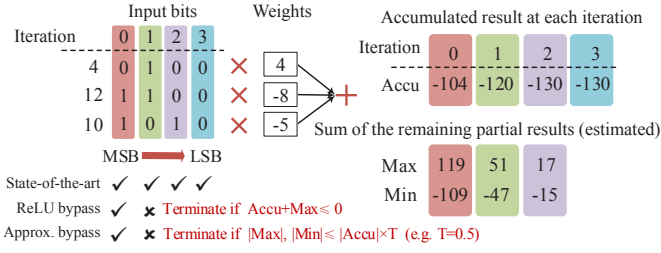


Fig. 2. Computation Reduction in the RRAM-based Accelerator

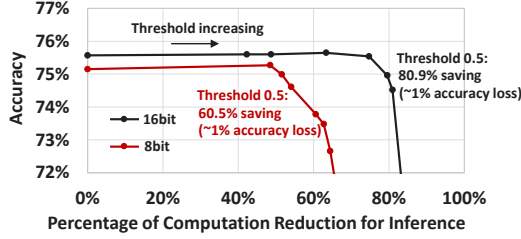


Fig. 3. Ideal Performance of the Adaptive Approximation for the 16-bit and 8-bit Implementations

maining iterations are considered to be redundant based on the possible values of their partial results estimated beforehand. Specifically, two schemes are exploited to identify the redundant iterations, as described below.

1) *ReLU-based Computation Bypass*: For an early identification of the negative outputs followed by ReLU, the maximum value of the sum of the partial results for the remaining iterations is estimated beforehand and represented as **Max** in Fig.2. (The method for estimating **Max** will be elaborated later.) If at any iteration, the sum of **Accu** and **Max** is not larger than zero ( $Accu + Max \leq 0$ ), the final output is considered to be non-positive and hence the MAC can be terminated in advance. Otherwise, the MAC will continue. For instance, the MAC in Fig.2 will be terminated after the second iteration since  $-120 + 51 \leq 0$ . It will be shown in the experimental results that the negative outputs of the CONV layers account for 57.5% of the total computations in the CifarQuick Model [8] on Cifar-10 [10], and 71.5% of their computations can be bypassed based on the estimation.

2) *Adaptive Approximation*: For the activations not supported by the ReLU-based bypass, adaptive approximation is proposed for the early termination of MAC. In general, the resilience of network allows the activations to deviate from their actual values within a certain range without affecting the classification result. Based on this, an adaptive approximation scheme is proposed as shown in Fig.2. If the magnitude of **Max** and **Min** (Min: minimum value of the sum of the remaining partial results) is not larger than a certain threshold (**T**) of the magnitude of **Accu** ( $|Max|, |Min| \leq |Accu| \times T$ ), which reflects the allowable deviation from the actual result, the MAC can be terminated. For instance, the last two iterations can be bypassed if **T** is set as 0.5 in Fig.2. The allowable deviation for triggering the bypass varies adaptively with  $|Accu|$ . The larger the  $|Accu|$ , the larger the allowable deviation is. **T** is an empirical tunable parameter to balance the accuracy and the complexity saving. A larger amount of computation reduction can be achieved by increasing **T**, but the accuracy loss will also increase at the same time. To obtain the ideal performance, i.e. the upper bound of complexity saving of the adaptive approximation, we assume we know the exact value of the output activation beforehand and so the actual partial result at each iteration is used instead of the estimated values. Based on this, we can exactly know which iterations will not be needed and the ideal maximum amount of bypass can be obtained. This ideal performance and the upper bound of saving at different **T** values for the general 16-bit fixed-point imple-

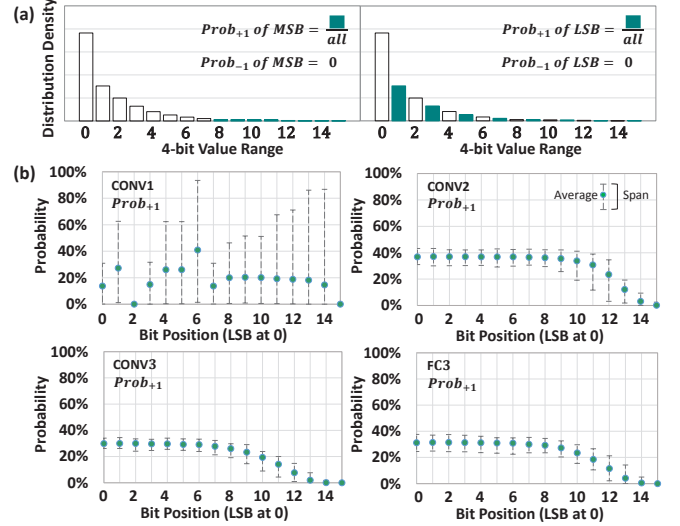


Fig. 4. (a) An Example for Extracting the Probabilities from the Distribution of the 4-bit Input Activations; (b) Average Probabilities and the Spans Extracted for the 16-bit implementation of CifarQuick.

mentation of CifarQuick is shown in Fig.3. A sweet spot is observed where 80.9% computations can be reduced with an accuracy loss of 1% for an optimum threshold value. Similarly, for the 8-bit fixed-point implementation, ideally a computation reduction of 60.5% can be achieved by the adaptive approximation at the same threshold as shown in Fig.3. The savings shown in Fig.3 assume a perfect knowledge of the partial result at each iteration. However in real situation, the actual partial results will not be known beforehand. Therefore we propose a method to get an accurate estimation on the maximum and minimum values of the partial result at each iteration.

## B. Making a Runtime Estimation on the Output Activation

Considering the MAC operation for an output activation, the worst-case maximum value of each partial result can be estimated by assuming all the input bit-weight products to be accumulated are as large as possible. In this case, for the layers with both positive and negative inputs, the maximum value of the partial result is equal to  $\sum |w| \times 2^i$  where  $w$  represents the weight in the kernel and  $i$  is the position of the input bit from bit-width-1 to 0. Similarly, the worst-case minimum value of each partial result is equal to  $-\sum |w| \times 2^i$ . For the hidden layers following ReLU, since there is no negative input, the maximum and minimum values of the partial result become  $\sum w_+ \times 2^i$  and  $\sum w_- \times 2^i$ , where  $w_+$  and  $w_-$  represent the positive and the negative weights, respectively. The computation bypass based on this loose bound of estimation ensures no accuracy loss, but the amount of the complexity reduction is small since the worst-case estimated values usually have much larger magnitude than the actual result due to several reasons. First, since the multi-bit input activation has been broken into multiple iterations, it is highly likely to have some input bits equal to zero even when the activation is positive. Thus, a considerable portion of input bit-weight products are actually zero. Moreover, in the worst case, all the input bit-weight products to be accumulated are assumed to have the same sign. However, in practice each non-zero product can either have positive impact or negative impact on the partial result.

To have a more practical estimation, a tighter bound based on the actual input activation statistics is proposed. Before the classification, the empirical activation statistics of each layer are obtained from the training images, and the probabilities of the input bit at each iteration to be +1 and -1 are calculated accordingly. As an example, Fig.4(a) illustrates how to extract the corresponding probabilities for the MSB and LSB from the distribution of the 4-bit input activations

in a CONV layer of CifarQuick. Specifically, the probability of MSB to be +1 is equal to the occurrence probability of the activation that has a value larger than 8. It can be easily extended to the implementations with different bit-widths. For a larger bit-width such as 16-bit, the activations will be partitioned into smaller bins. The average value and the span of the probability of each input bit to be +1 extracted for the 16-bit implementation of CifarQuick are shown in Fig.4(b). For the hidden layers, the probability of the MSB to be +1 is small since most of the activations have small values. For the LSB, the probability is within the range from 25% to 40% since a large portion of input activations are zero. The probability of having a -1 input is zero for the layers following ReLU. CONV1 is different from others due to the mean subtraction for image pre-processing. It is worth noting that the activation distribution normally doesn't change much for different images in the dataset, and thus the empirical probabilities can be generally utilized for the estimation. The maximum and minimum values of the partial result estimated at a specific iteration  $i$  ( $i=0$  for LSB) are given by:

$$\begin{aligned}
max &= (max_+ + max_-) \times 2^i, min = (min_+ + min_-) \times 2^i \\
max_+ &= \sum w_+ \times Prob_{+1,max} + \sum |w_-| \times Prob_{-1,max} \\
max_- &= \sum -w_+ \times Prob_{-1,min} + \sum w_- \times Prob_{+1,min} \\
min_+ &= \sum w_+ \times Prob_{+1,min} + \sum |w_-| \times Prob_{-1,min} \\
min_- &= \sum -w_+ \times Prob_{-1,max} + \sum w_- \times Prob_{+1,max}
\end{aligned} \quad (3)$$

where  $max$  and  $min$  represent the estimated maximum and minimum values of the partial result, respectively. To estimate  $max$ , the input bit-weight products with different signs need to be separately considered. We first consider the case where the input bits and the corresponding weights have the same signs to estimate the sum of the positive products ( $max_+$ ).  $\sum w_+$  and  $\sum w_-$  represent the sums of the positive weights and the negative weights in the kernel, respectively. For a better accuracy, a conservative bound should be used for the estimation, and thus we use  $Prob_{+1,max}$  and  $Prob_{-1,max}$  to estimate  $max_+$  where  $Prob_{+1,max}$  and  $Prob_{-1,max}$  represent the maximum probabilities of the input bit to be +1 and -1, respectively. Also we need to estimate the sum of the negative products ( $max_-$ ) when the input bits and weights are of opposite signs. Again to have a conservative bound we use  $Prob_{+1,min}$  and  $Prob_{-1,min}$  which represent the minimum probabilities of the input bit to be +1 and -1, respectively. The minimum value of the partial result ( $min$ ) can be estimated in a similar way. This statistics-based estimation is more precise than the worst-case estimation.

### C. Hardware Architecture

The analog computation is done inside the in-situ processing units (IPUs) similar as Fig.1(b). Each IPU contains a group of 1-bit DACs at the input of the wordlines, a pair of differential RRAM crossbars to store the positive and negative weights, respectively, the sample-hold units to hold the bitline currents, a single ADC which is time-shared by the bitlines, and a shift-add unit to aggregate the partial results after ADC for the MAC operation. In order to make a comparison with the state-of-the-art RRAM-based accelerator, we adopt a hierarchical architecture similar to ISAAC presented in [17] as the baseline, and compare the proposed CompRRAE with ISAAC in terms of the energy efficiency, throughput, and area cost. Similar to ISAAC, each accelerator contains multiple tiles connected with a concentrated-mesh at the top level. The architecture inside a tile is shown in Fig.5(a). Each tile contains multiple in-situ multiply-accumulate modules (IMAs) sharing the same centralized memories

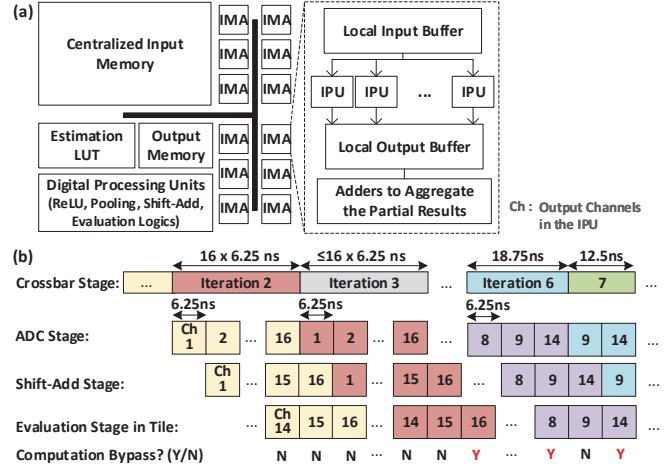


Fig. 5. (a) Hardware Architecture inside a Tile; (b) The Pipeline of CompRRAE.

and digital processing units which are used to execute digital operations such as shift-add, ReLU and pooling. Inside the tile, the IMAs are connected through a shared bus. Inside each IMA, there are multiple IPUs sharing a local input buffer and a local output buffer which hold the input and output activations, respectively, during the MAC.

In order to implement the proposed runtime estimation, first, the probabilities are extracted offline. Based on Eq.(3), the estimated maximum and minimum values of the partial result at each iteration are computed independently for each output channel. Then, the corresponding estimated partial results are summed up to get the **Max** and **Min** for each iteration. Same as the example shown in Fig.2, there are totally  $N-1$  **Max** and **Min** values for each output channel, where  $N$  is the number of iterations, i.e. the bit-width of the activation-1. This process is done offline and the estimated **Max** and **Min** values are stored in a look-up table (LUT) in the tile. During runtime, at each iteration, the estimated value of the output activation is the sum of the actual accumulated result and the estimated value for the remaining iterations read from the LUT. The evaluation logics to compute the formulas in Fig.2 for deciding whether to skip the remaining iterations include an adder for the ReLU-based bypass and a multiplier and comparator for the approximation-based bypass. The size of the tile is normally large enough for mapping a complete kernel of the CONV layers. However, each kernel may occupy multiple IMAs and thus the local results in the IMAs are required to be sent out through the shared bus and aggregated in the tile. To minimize the overhead of data transfer, each kernel is preferred to fully occupy the IPUs in one IMA first before occupying others when mapping the network, and the calculated results are firstly aggregated locally inside the IMAs before sent out through the shared bus.

We use the same configuration as that in ISAAC where each memristor has 2-bit precision in the  $128 \times 128$  RRAM crossbar. Since each 16-bit weight takes 8 memristors to store, there are 16 output channels mapped to one IPU. The pipeline schedule of CompRRAE is shown in Fig.5(b). After finishing each crossbar computation, the bitline results are firstly latched in the sample-hold circuits. In the next stage, a 1.28GHz ADC sequentially converts the 8 bitline currents for an output activation in 6.25ns in the IPU. The 8 bitline results are processed by the shift-add to generate a local partial result in the IPU in the next 6.25ns. Then, in the next stage, the local partial results of different IMAs are aggregated in the tile and update the intermediate accumulated result of the MAC. This result together with the estimated values stored in the LUT will be sent to the evaluation logics to decide whether the computation can be terminated earlier based on the algorithms presented in Section IV, and the control signals will be generated and sent back to the IMAs. All these operations will be

finished in the 6.25ns time frame. At the beginning of the MAC operation, i.e. the first iteration, it takes  $16 \times 6.25\text{ns}$  to finish the iteration for the 16 output activations in the IPU. As the computation goes on, some of the output activations may be bypassed and the time for each iteration in the IPU may become shorter. After all the IPU's finish the computation, the results will be sent to the next layer, and the next MAC operation of the current layer will start. Since the iterations may take less time in CompRRAE due to the computation bypass, the bandwidth of the input memories to provide the necessary input activations to the IPU's have to be increased. If we need to maintain the input memory bandwidth the same as that in ISAAC, the number of IPU's in each tile needs to be reduced to make sure the memory bandwidth can support the IPU's which are now running faster. At the same time, since less IPU's are used in each tile, more tiles are needed for mapping the same network and this will cause area overhead.

Compared with ISAAC, the energy overhead mainly comes from the evaluation logics, the additional data transfers through the shared bus, and the extra memory accesses of the LUT and the centralized output buffer. Extra area overhead will be required for the evaluation logics and the LUT. The detailed analysis will be discussed in the next section.

## V. EXPERIMENTAL RESULT

### A. Models of Energy, Area, and Throughput

The operation parameters and the corresponding energy and area data for the major components of CompRRAE are summarized in Table I. All the memories and the shared buses are modeled at 32nm in CACTI6.5 [15]. The centralized input memory is implemented using eDRAM. The local buffers, the centralized output memory and the LUT are implemented using SRAM. The conductance range and the area of RRAM are taken from the Stanford-PKU RRAM model [9], and the corresponding power is simulated using a device-level simulator implemented in C++. The parameters of the 1-bit DAC are obtained through a real design implemented in Cadence at TSMC 65nm and scaled shown to 32nm process. Same as ISAAC, an 8-bit SAR ADC is adopted, and the power and area are taken from [12]. The evaluation logics are designed and implemented in Verilog and synthesized using TSMC 65nm. The power and area are obtained and scaled down to 32nm process. The parameters of other digital processing units such as the shift-add and the sample-hold are adapted from ISAAC [17]. The time for the IPU's to finish the MAC operation for each layer is used to model the execution time, and a simulation-based throughput model is built in SystemC. We also calculate the corresponding energy, throughput, and area of ISAAC as a baseline to compare with.

### B. Benchmarks

We use two benchmarks to compare CompRRAE with ISAAC. The first benchmark is the LeNet-5 [13] which has two CONV layers and two FC layers trained on the handwritten digit dataset MNIST. The second benchmark is the medium-sized CifarQuick model [8] with three CONV layers and two FC layers trained on the color image dataset Cifar-10 [10]. The proposed schemes are firstly tested for a 16-bit quantization for comparison with ISAAC, and then tested for an 8-bit quantization to demonstrate the effectiveness of CompRRAE under an aggressive quantization scheme. The accuracy of the fixed-point implementations are summarized in Table II.

### C. Results of the ReLU-based Computation Reduction

The negative output activations in the CONV layers of CifarQuick account for 57.5% of the total computations during the inference. On average, over 99.9% of the negative outputs are detected based on the

Table I. Power and Area Estimation

Centralized Memories and Buses			
Component	Spec	Energy( $nJ$ )	Area( $\mu m^2$ )
Input Memory (eDRAM)	size: 64KB bus width: 256bit	0.0188 (0.38mW leakage)	46000
Output Memory (SRAM)	size: 1KB bus width: 128bit	0.0008 (0.13mW leakage)	3900
Estimation LUT (SRAM)	size: 5KB bus width: 160bit	0.0035 (0.002mW leakage)	9600
Bus of Input Path	num: 256 delay: 0.44ns	0.0042	80000
Bus of Output Path	num: 128 delay: 0.43ns	0.0020	39100
Local Memories (Shared among 8 IPU's)			
Component	Spec	R/W Energy( $nJ$ )	Area( $\mu m^2$ )
Input Buffer	size: 2KB bus width: 256bit	0.0019 (0.42mW leakage)	7400
Output Buffer	size: 256B bus width: 128bit	0.0005 (0.05mW leakage)	2600
IPU Parameters at 1.28GHz (80 MACs per Tile)			
Component	Spec	Power( $mW$ )	Area( $\mu m^2$ )
DAC	resolution: 1 bit num: 256	0.25	668
ADC	resolution: 8 bit num: 1	3.1	1500
Memristor Crossbar	resolution: 2 bit num: 2	Cifar-10: 1.5 MNIST: 0.7	264
Sample-Hold	num: 128	0.001	5
Shift-Add	num: 1	0.05	60
Other Tile Parameters at 1.28GHz			
Component	Spec	Power( $mW$ )	Area( $\mu m^2$ )
Evaluation Logic	num: 8	0.79	320
Shift-Add	num: 8	0.4	480

Table II. Accuracy of the Fixed-Point Implementations

Benchmarks	16-bit Representation	8-bit Representation
CifarQuick	75.57%	75.15%
LeNet-5	99.13%	99.09%

runtime estimation and over 71.5% of the computations corresponding to these negative outputs are reduced for the 16-bit implementation. For the 8-bit implementation, over 98.3% of the negative outputs are detected and 44.1% of their computations are reduced. Thus, the overall ReLU-based computation reduction for a complete inference are 40.2% and 23.8% for the 16-bit and 8-bit implementations, respectively. No accuracy has been compromised for both implementations. Since the CONV layers of LeNet-5 are not followed by ReLU, the computation will only be reduced by the adaptive approximation. The performance of the ReLU-based computation bypass in CifarQuick is summarized in Table III.

### D. Results of the Adaptive Approximation

The results of the computation bypass based on the adaptive approximation are shown in Fig.6. To maximize the amount of computation reduction while maintaining a high accuracy, the optimal threshold for the approximation is empirically found as 0.8 for the 16-bit implementation of CifarQuick, where 67.4% of computations

Table III. Results of the ReLU-based Computation Reduction in Cifar-Quick

Computation Reduction	16-bit Implementation	8-bit Implementation
For the Negative Outputs followed by ReLU	71.5%	44.1%
For a Complete Inference	40.2%	23.8%



Table IV. The Overall Computation Reduction, Energy Efficiency, Throughput, and Area Efficiency

Performance	16-bit CifarQuick		8-bit CifarQuick		16-bit LeNet-5		8-bit LeNet-5	
	Baseline	CompRRAE	Baseline	CompRRAE	Baseline	CompRRAE	Baseline	CompRRAE
Accuracy	75.57%	75.44%	75.15%	75.00%	99.13%	98.97%	99.09%	98.90%
Computation Reduction for a Complete Inference	-	69.4%	-	39.1%	-	78.5%	-	45.4%
Energy Consumption ( $mJ/frame$ )	5.80e-2	2.00e-2	1.41e-2	1.00e-2	1.71e-2	5.65e-3	4.31e-3	2.61e-3
Energy Efficiency ( $frames/J$ )	1.72e+4	5.00e+4	7.10e+4	1.00e+5	5.83e+4	1.77e+5	2.32e+5	3.83e+5
Throughput ( $frames/s$ )	603.0	1715.6	1205.2	1978.3	1082.0	4897.1	2158.0	4163.1
Area ( $mm^2$ )	0.5125	0.5816	0.3022	0.3468	1.5375	1.7252	0.8204	0.9243
Area Efficiency ( $frames/s/mm^2$ )	1176.8	2949.8	3988.1	5704.4	703.7	2838.5	2630.4	4504.0

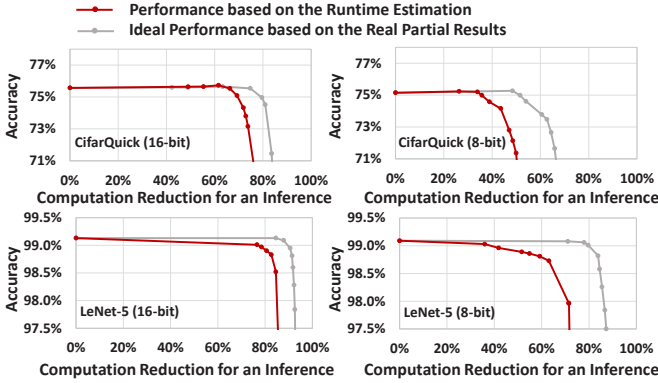


Fig. 6. Results of the Adaptive Approximation

can be reduced for the inference with an accuracy loss as small as 0.13%. For the 8-bit implementation of CifarQuick, 35.8% computation reduction is achieved at the same threshold for the inference with only 0.16% accuracy loss. Similar trend has been observed in LeNet-5, where 78.5% and 45.4% computation reductions are obtained for the 16-bit and 8-bit implementations, respectively. The accuracy loss is smaller than 0.19%.

#### E. Overall Performance and Overhead Analysis

The overall computation reduction, energy efficiency improvement, and throughput improvement after combining the two proposed schemes are summarized in Table IV. The overall computation reduction achieved for the 16-bit implementation of CifarQuick is 69.4% with 0.13% induced accuracy loss. Therefore, the energy efficiency and throughput are improved by 2.9 times and 2.8 times, respectively. The energy overhead caused by the runtime estimation (i.e. the evaluation logics, the extra data transfers through the shared bus, and the extra memory accesses) accounts for 3.4% of the overall energy consumption. Compared with ISAAC, there is a 13.5% area overhead due to the estimation logics, the LUT, and the extra tiles occupied. However, due to the improvement of throughput, the area efficiency is improved by 2.5 times. For the 8-bit implementation of CifarQuick, 39.1% of computations are reduced. Thus the energy efficiency and throughput are improved by 1.4 times and 1.6 times, respectively, at a cost of 0.15% accuracy loss and 14.8% area overhead. Similar results have been observed for LeNet-5. The improvements in energy efficiency and throughput are 3.0 times and 4.5 times for the 16-bit implementation. For the 8-bit implementation, the corresponding improvements are 1.6 and 1.9 times, respectively. Around 12% area overhead is induced for the implementations of LeNet-5.

#### VI. CONCLUSIONS

In this paper, a RRAM-based CNN accelerator is proposed to reduce the computations during the inference. The computations are reduced by exploiting the output sparsity and the adaptive approximation based on the runtime estimation on the maximum and min-

imum values of the output activation. It is implemented under different quantization schemes, and the corresponding energy efficiency and throughput are significantly improved.

#### REFERENCES

- [1] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh. Snapea: Predictive early activation for reducing computation in deep convolutional neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 662–673, June 2018.
- [2] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. *SIGARCH Comput. Archit. News*, 44(3):1–13, June 2016.
- [3] X. Chen, J. Jiang, J. Zhu, and C.-Y. Tsui. A high-throughput and energy-efficient rram-based convolutional neural network using data encoding and dynamic quantization. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference, ASPDAC '18*, pages 123–128, Piscataway, NJ, USA, 2018. IEEE Press.
- [4] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-47*, pages 609–622, Washington, DC, USA, 2014. IEEE Computer Society.
- [5] G. E. Dahl, T. N. Sainath, and G. E. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8609–8613, May 2013.
- [6] B. Feinberg, S. Wang, and E. Ipek. Making memristive neural network accelerators reliable. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 52–65, Feb 2018.
- [7] H. Ji, L. Song, L. Jiang, H. Li, and Y. Chen. Recom: An efficient resistive accelerator for compressed deep neural networks. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 237–240, March 2018.
- [8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia, MM '14*, pages 675–678, New York, NY, USA, 2014. ACM.
- [9] Z. Jiang and H.-S. P. Wong. Stanford university resistive-switching random access memory (rram) verilog-a model, Oct 2014.
- [10] A. Krizhevsky and G. E. Hinton. Learning multiple layers of features from tiny images. 1, 01 2009.
- [11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, pages 1097–1105,

USA, 2012. Curran Associates Inc.

- [12] L. Kull, T. Toifl, M. L. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. A. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici. A 3.1mw 8b 1.2gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32nm digital soi cmos. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 468–469, Feb 2013.
- [13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.
- [14] B. Moons, B. D. Brabandere, L. J. V. Gool, and M. Verhelst. Energy-efficient convnets through approximate computing. *CoRR*, abs/1603.06777, 2016.
- [15] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 40*, pages 3–14, Washington, DC, USA, 2007. IEEE Computer Society.
- [16] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. *CoRR*, abs/1510.07945, 2015.
- [17] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture, ISCA '16*, pages 14–26, Piscataway, NJ, USA, 2016. IEEE Press.
- [18] Y. Wang, B. Li, R. Luo, Y. Chen, N. Xu, and H. Yang. Energy efficient neural networks for big data analytics. In *Proceedings of the Conference on Design, Automation & Test in Europe, DATE '14*, pages 345:1–345:2, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association.
- [19] J. Zhu, J. Jiang, X. Chen, and C.-Y. Tsui. Sparsenn: An energy-efficient neural network accelerator exploiting input and output sparsity. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 241–244, March 2018.