Master's Thesis

Architecture-Accuracy Co-optimization of ReRAM-based Low-cost Neural Network Processor

Se-Gi LEE

Department of Electrical Engineering

Graduate School of UNIST

2020

Architecture-Accuracy Co-optimization of ReRAM-based Low-cost Neural Network Processor

Se-Gi Lee

Department of Electrical Engineering

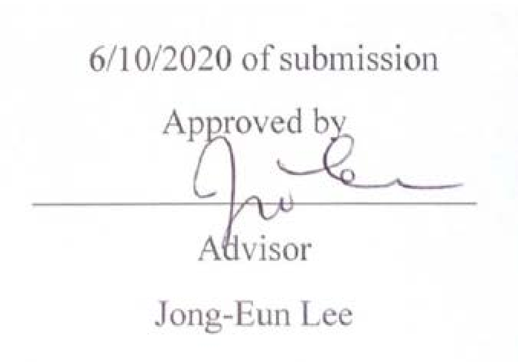
Graduate School of UNIST

UNIST
ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

Architecture-Accuracy Co-optimization of ReRAM-based Low-cost Neural Network Processor

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Se-Gi Lee



6/10/2020 of submission
Approved by

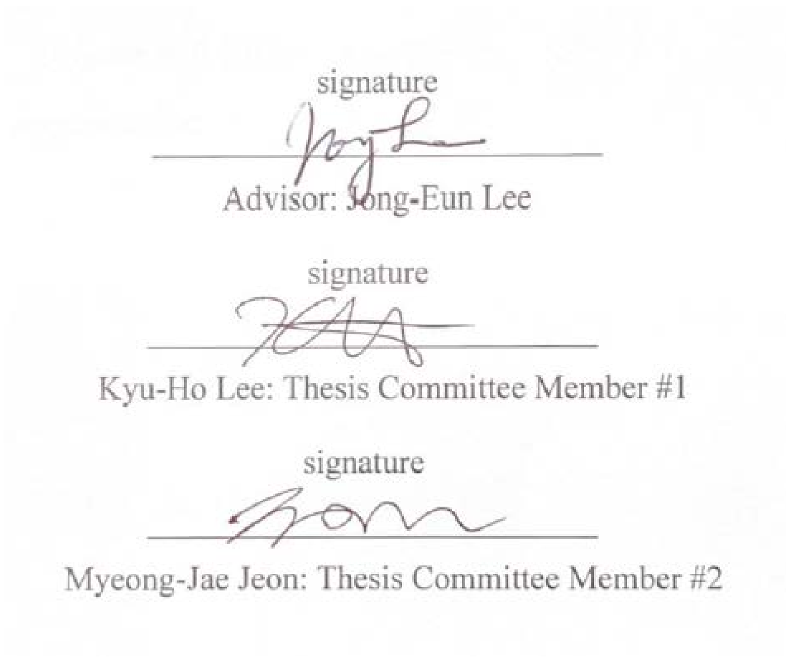
Advisor
Jong-Eun Lee

Architecture-Accuracy Co-optimization of ReRAM-based Low-cost Neural Network Processor

Se-Gi Lee

This certifies that the thesis of Se-Gi Lee is approved.

6/10/2020 of submission



signature

Advisor: Jong-Eun Lee

signature

Kyu-Ho Lee: Thesis Committee Member #1

signature

Myeong-Jae Jeon: Thesis Committee Member #2

Abstract

Resistive RAM (ReRAM) is a promising technology with such advantages as small device size and in-memory-computing capability. However, designing optimal AI processors based on ReRAMs is challenging due to the limited precision, and the complex interplay between quality of result and hardware efficiency. In this paper we present a study targeting a low-power low-cost image classification application. We discover that the trade-off between accuracy and hardware efficiency in ReRAM-based hardware is not obvious and even surprising, and our solution developed for a recently fabricated ReRAM device achieves both the state-of-the-art efficiency and empirical assurance on the high quality of result.

Contents

List of Figures

List of Tables

Chapter I

INTRODUCTION

Recently the resistive RAM (ReRAM) technology is getting much attention [22, 23] due to many advantages including small device size, non-volatility, and compatibility with the CMOS technology. Also some devices show very high on-off ratio (Roff/Ron) [17], enabling in-memory computing, which is crucial to overcoming the von Neumann bottleneck and the memory wall problem in computer architecture [24].

In-memory computing with ReRAM is realized by arranging a number of ReRAM cells in a 2D crossbar array structure and activating all the rows (wordlines) simultaneously, which produces the result of matrix-vector multiplication (MVM) computation at bitlines, performing O(n2) computation in a single cycle. The MVM operation is the main computation kernel in neural network applications, and there has been much previous work on ReRAM-based neural network hardware, suggesting order-of-magnitude improvement in energy efficiency [5, 7, 21–23].

However, when it comes to actual hardware implementation, there is very little work reported, and the reported accuracy is generally low as well (around 91% for the MNIST dataset) [15, 18]. There are many reasons for that but an important one is the programming difficulty: programming ReRAM devices in a crossbar array is extremely difficult due to write disturbance, limited endurance, and long write time [27]. Consequently, to avoid frequent re-programming of ReRAM cells at runtime, ReRAM-based neural networks unfold all synapses (= ReRAM cells) spatially [5, 7, 21], resulting in a fully-parallel architecture, i.e., all neurons/synapses are implemented using dedicated resources. Not only that, but all the layers of a network should be unrolled in a fully parallel fashion.

Thus from the system-level design perspective, designing ReRAM based full-network hardware using today’s technology involves two key problems: (i) how to optimize system performance (i.e., classification accuracy) while meeting stringent area constraints, and (ii) how to efficiently and effectively implement all types of layers, especially normalization layers. The first challenge may appear similar to that of digital CMOS-based neural networks, but is in fact very different, because unlike digital CMOS-based neural networks, which have the freedom of trading between area and delay, ReRAM-based neural networks cannot trade off delay for smaller area. To reduce area, one must resort to algorithmic optimizations such as reducing network size and precision, which are bound to affect accuracy. This calls for complex algorithm-architecture co-design, which is lacking in the current literature.

At the same time, to maintain high accuracy in low-precision networks batch normalization layers are found to be crucial [6]. But since a straightforward implementation of batch normalization is quite expensive especially compared to ReRAM-based layers, it is worthwhile to explore alternative methods.

In this paper we present an accuracy-aware design optimization study for a ReRAM-based neural network, targeting low-cost, low-power applications. We explore network size, weight precision, activation precision, and various ways to implement batch normalization, and arrive at an optimal design in terms of area and accuracy. Compared with previous work on ReRAM-based neural networks, whereas the previous work emphasizes on architectural scalability [5, 7, 21], our optimization considers both accuracy and efficiency, and is also readily implementable using today's technology. Though we do not consider other ReRAM-specific issues such as read disturbance and device variability, they have previously known solutions (e.g., [8]), and are orthogonal to the issues considered in this paper.

In this paper we make the following contributions. First we present an algorithm-architecture co-optimization study targeting a low-power, low-cost image classification application scenario such as in IoT (internet-of-things) devices, optimizing both accuracy and hardware complexity together. Second we analyze efficient ways to implement batch normalization for analog-based neural networks, including a novel scheme that is hardware-friendly. Finally, based on the previous work and our own circuit implementation, we estimate that our architecture can deliver extremely high efficiency (361.02 TOPS/mm2 and 589.57 TOPS/W) at acceptable accuracy (about 95.97% for MNIST in simulation, not considering ReRAM nonidealities).

Chapter II

RELATED WORK

2.1 ReRAM-based Neural Network Hardware

Most of the previous work on ReRAM-based neural network hardware falls into the category of architecture study focusing on hardware efficiency (e.g., TOPS/W), without much attention to the functional correctness of such systems beyond simple noise analysis (e.g., [21]). In ISAAC [21], activation and weights are represented in 16-bit digital signals, which should be sufficient for inference but require ADC/DAC. To minimize the overhead of ADC/DAC, ISAAC processes digital activation in a bit-serial manner, and analog-todigital conversion, which is at the output of a crossbar array, is done by a single ADC in a time-multiplexed manner, which limits computation throughput while saving area. Even then, ADC/DAC account for 23.06% of area while ReRAM crossbar takes only 0.47%. This shows a potential for higher efficiency of co-optimizing precision and area.

BISAAC [7] reduces the overhead of peripheral circuit by executing XNOR operation, achieving about 2.7× improvement in power consumption over ISAAC. However, all the above work considers hardware efficiency only, without optimizing the neural network algorithm (e.g., using hardware-friendly binary or ternary neural networks).

There are a few papers reporting ReRAM-based deep neural network (DNN) chips that have been actually built, but in those cases the reported DNN accuracy is universally low. A ReRAM-based DNN was designed and fabricated [18], targeting MNIST with three fully connected layers. While its energy efficiency is very high (66.5 TOPS/W), the accuracy is less than 91%. Another ReRAM-based neural network chip [15] targeting MNIST consists of two fully-connected layers, and despite the newly endowed on-chip learning capability, its accuracy is under 92%.

On the other hand, very impressive accuracy of 98.8% for MNIST and 88.52% for CIFAR-10 has been reported [2, 25], but the hardware consists of a ReRAM memory only while the rest of DNN computation, even including the subtraction operation between the positive-weight and negative-weight crossbar outputs, is implemented on an FPGA and a host computer. On top of that, in this work the crossbar columns are accessed sequentially, in a way analogous to ISAAC, to minimize the sneakpath problem at the expense of much reduced computation throughput, which cannot realize the full potential of a crossbar architecture.

2.2 Low-Precision DNN

In this paper precision refers to the number of quantization levels used for a certain variable, such as weight and activation, in a quantized neural network. Previous work on neural network quantization shows that it is possible to achieve near-baseline performance if we binarize only either activation or weights of a neural network (but not both), where the baseline is one that uses floating-point for both [4, 9, 16]. For instance, using ternary or binary weights with floating point activation is shown to give 84～81% top-5 accuracy on ImageNet [20] classification [16], which is close to the 86.76% baseline. However, binarizing both activation and weights [6], which is the most hardware-friendly version, seems to work only for mid-to-small datasets such as MNIST [13] and CIFAR-10 [11]. Also the authors [6] stress that having batch normalization layers is crucial to achieving high accuracy for low-precision networks, which agrees with our experimental results.

Chapter III

DESIGN FRAMEWORK CONSIDERING BOTH ACCURACY AND EFFICIENCY

3.1 System Architecture

To illustrate our design framework we consider a small, yet highly efficient neural network processor for extremely cost-sensitive applications such as IoT devices. For dataset, we use the MNIST handwritten digit recognition dataset [13]. An analysis of the previous work [7, 21] suggests that ADC/DAC consumes most of the area/power budget even when ADC is shared across all columns of an ReRAM crossbar array. This motivates the use of analog activation, which can eliminate the ADC/DAC circuitry between layers. While using digital activation may be necessary to support very large DNN models and complex datasets, not all applications must support the largest datasets, and certainly not ours targeting IoT devices.

As mentioned earlier, all layers and all neurons are implemented with dedicated resources, and the output of the first layer is fed to the second layer, and so on, possibly with digital or analog buffers inserted between layers to maximize throughput, as illustrated in Figure 1.

To maximize area efficiency, we must minimize area while not sacrificing inference accuracy.

In our design, area is directly proportional to the number of neurons and synaptic weights. Therefore, we explore different input sizes (e.g., using image cropping) as well as different activation/weight precisions. However, there are also normalization layers such as batch normalization (BN) [10] and local response normalization (LRN) [12] that are shown to be essential for high accuracy in addition to fast training time. How to handle 2 those layers strongly impacts the accuracy and cost of the hardware implementation. Thus we next discuss BN, which is popular among extremely low precision networks [6].

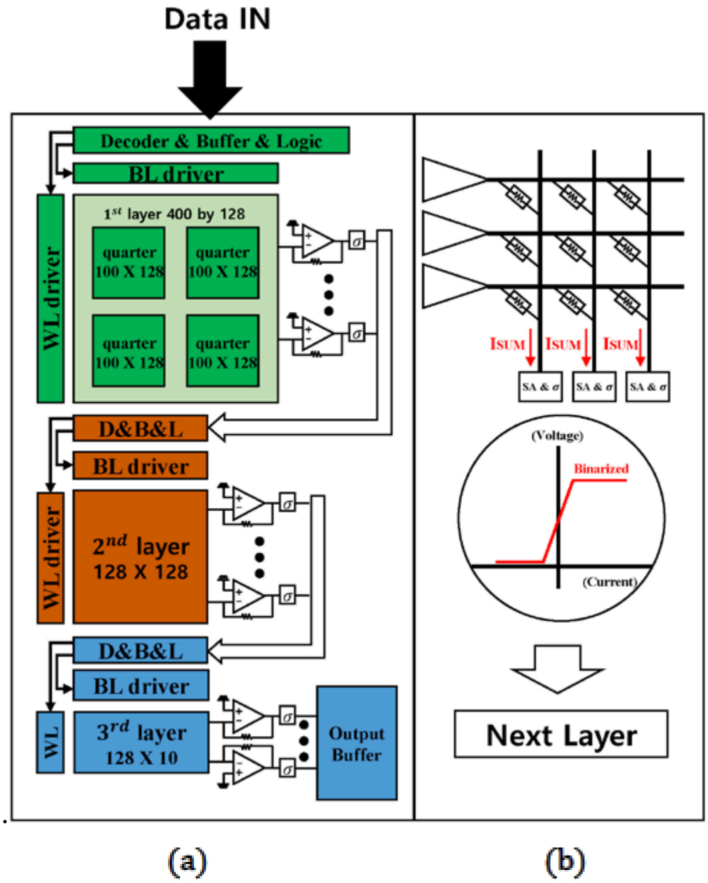


Figure 1. System overview. (a) Top-level system architecture where the priority encoder outputs class label. (b) A layer is implemented by a ReRAM crossbar array and I-V converters(realizing activation function).

3.2 Batch Normalization and Activation Quantization

A batch normalization layer is defined as follows, where y_j is the j^{th}-channel's output of the preceding layer, μ_j and σ_j are the mean and variance of y_j across the batch, and γ_j and β_j are the trainable parameters of BN.

$$\mathrm{BN}(y_j) = \frac{y_j - \mu_j}{\sigma_j}\gamma_j + \beta_j \tag{1}$$

During inference, the batch size is often very small (e.g., 1), in which case a pre-computed version of μ_j and σ_j (computed across all training data) is used. Therefore, we can merge BN into the preceding layer as follows (Note: BN is applied before activation function). For simplicity, we assume that the preceding layer is a fully-connected layer, but it can be merged equally well to convolution layers: $y_j = \sum_i x_i w_{ij} + b_j$.

$$\mathrm{BN}(y_j) = \frac{\gamma_j}{\sigma_j}\left(y_j + \left(-\mu_j + \frac{\sigma_j}{\gamma_j}\beta_j\right)\right) \tag{2}$$

$$= \sum_i x_i w'_{ij} + b'_j \tag{3}$$

The above holds true if we define w'_{ij} and b'_j as follows.

$$w'_{ij} = \frac{\gamma_j}{\sigma_j} w_{ij}\,, \tag{4}$$

$$b'_j = \frac{\gamma_j}{\sigma_j}(b_j + \left(-\mu_j + \frac{\sigma_j}{\gamma_j}\beta_j\right)) \tag{5}$$

This way, we can merge a batch normalization layer into the preceding fully-connected or convolution layer for inference.

It is important to understand that when quantizing a network, we must quantize the original weight w_{ij} and not w'_{ij}, in order to reap the benefit of BN. Unfortunately, this means that the above merging method makes quantized weights no longer quantized after merging. For instance, if w_{ij} is binary ({-1, +1}), w'_{ij} may not be binary, and may require very high precision. While the above merging method may still be used (with some error) for high-precision weights, for low-precision weights (e.g., binary weights) much larger error can be introduced, except for one case: binary activation case.

3.2.1 Binary Activation. If activation is binary, the activation function is the sign function. Hence, we can ignore magnitude, and rewrite y'_j(activation output after BN) as follows [26]. This is an exact method that can merge BN into the preceding layer.

$$w'_{ij} = \mathrm{sign}(\frac{\gamma_j}{\sigma_j})w_{ij}, \tag{6}$$

$$b'_j = \mathrm{sign}(\frac{\gamma_j}{\sigma_j})(b_j + \left(-\mu_j + \frac{\sigma_j}{\gamma_j}\beta_j\right)) \tag{7}$$

3.2.2 Multi-bit Activation and SBN. If activation is non-binary, one possible workaround (to keep weights quantized) is to merge BN into the following activation function. For instance, a scaling parameter, $\frac{\gamma_j}{\sigma_j}$, can be multiplied into the domain of the activation function, which gives mathematically identical result. The problem with this method is that the scaling parameter differs across channels, leading to very complex hardware design.

As an approximate method, we propose shared-parameter BN(SBN), which is to share the scaling parameter $\frac{\gamma_j}{\sigma_j}$ across all channels, to simplify hardware implementation. However, since this reduces the number of BN parameters, performance loss may be inevitable. Therefore, when deciding the precision for activation, all the above issues must be taken into account to get the best of both accuracy and hardware efficiency.

3.3 Weight Quantization

There are three factors affecting the decision of the weight quantization. First, the network itself dictates the minimum precision for weights to guarantee a certain level of inference accuracy. Second, multi-bit ReRAM cells [1] can naturally represent multi-bit weight values. Third, if the precision required by the algorithm is higher than that of ReRAM cells, multiple ReRAM cells can be used together to represent a single weight value, which incurs additional hardware overhead such as digital shifter-adder [21] or an analog equivalent to combine the results from multiple bitlines.

Chapter IV

ALGORITHM-ARCHITECTURE CO-OPTIMIZATION

We explore various network designs evaluating their performance. Network performance is reported in terms of test accuracy. Our model is based on the MNIST DNN of the BinaryNet framework,1 which uses binary values for both activation ({0, 1}) and weight ({−1, 1}), but we explore different precision and input sizes in a bid to strike a better balance between hardware efficiency and accuracy. The BinaryNet for MNIST consists of fully-connected layers only (along with batch normalization), but for other datasets such as CIFAR-10 and SVHN, BinaryNet uses convolutional neural networks (CNNs), to which our methodology is equally applicable. For neural network training, we use the Torch7 framework with the default setting, which is extended to support our design exploration.

The final architecture that gives the best tradeoff between accuracy and area, turns out to be a binarized network with this configuration: 400-128-128-10. The input size is cropped to 20x20 (=400), and the two hidden layers have 128 neurons each. The precision of both weights and activations is determined to be 1-bit, which allows for the use of a simpler BN implementation. In the following, due to space limitation we give a justification of the final architecture instead of the entire design exploration, which has a combinatorial complexity.

4.1 Effect of Input Cropping

Reducing the input size is primarily motivated by the I/O pin restriction. Moreover, reducing the input size can have a huge impact on the total number of synaptic weights. Input cropping is preferred to image scaling due to its cheaper computation (we binarized the input). Table 1 summarizes the result (i.e., test accuracy) of input cropping, where the rest of the network consists of 2 fully-connected layers with 128 neurons each. Our training result shows that there is virtually no degradation until 20 × 20 compared to the original size, 28 × 28. Cropping further degrades accuracy significantly, over 5 percent point (%p) at 12 × 12 image. Due to the relatively large size of the input layer in our DNN, input cropping contributes to about 42% reduction of the network size.

Table 1. Input size exploration (hidden size:128,128)

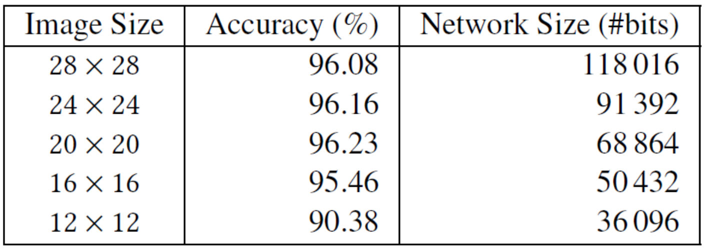


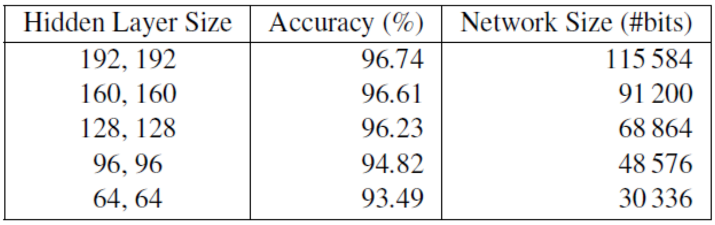
Image Size	Accuracy (%)	Network Size (#bits)
28 × 28	96.08	118 016
24 × 24	96.16	91 392
20 × 20	96.23	68 864
16 × 16	95.46	50 432
12 × 12	90.38	36 096

4.2 Exploring Hidden Layer Size

Next we consider the effect of different hidden layer sizes (we vary both the layer sizes together). As compared in Table 2, increasing layer size gives better result in general but also increases the cost. Unlike the input size which is only related to the first layer, the hidden layer size affects every layer, actually quadratically in middle layers. Considering both accuracy and cost, the table shows that using

128 neurons is the best.

Table 2. Hidden layer size exploration (input size: 20x20)

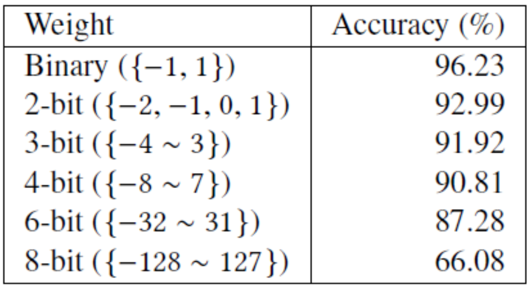


Hidden Layer Size	Accuracy (%)	Network Size (#bits)
192, 192	96.74	115 584
160, 160	96.61	91 200
128, 128	96.23	68 864
96, 96	94.82	48 576
64, 64	93.49	30 336

4.3 Exploring Weight Precision

Unlike network size, changing weight precision has deeper consequences. Everything else being equal, higher precision should produce better accuracy, but it would not only increase the number of ReRAM cells required, but increase the overhead of periphery. For example, if multiple ReRAM cells are needed to represent a weight value, we need to combine the results from multiple bitlines, such as using shifter-adder or its analog equivalent. Therefore, in this exploration we vary both weight precision and the number of weight parameters at the same time such that the total number of weight bits remain roughly the same. Table 3 summarizes the result. The batch normalization layer is implemented in software. Our result suggests that when the network size is fixed, using 1-bit weights gives the highest accuracy.

Table 3. Weight precision exploration (activation: 1-bit, # of neurons is adjusted to make total # of synaptic weight bits constant)



Weight	Accuracy (%)
Binary ($\{-1, 1\}$)	96.23
2-bit ($\{-2, -1, 0, 1\}$)	92.99
3-bit ($\{-4 \sim 3\}$)	91.92
4-bit ($\{-8 \sim 7\}$)	90.81
6-bit ($\{-32 \sim 31\}$)	87.28
8-bit ($\{-128 \sim 127\}$)	66.08

4.4 Exploring Activation Precision and BN

Table 4 shows how multi-bit activation and batch normalization affect performance (i.e., inference accuracy). Without BN, networks fail to train regardless of the activation precision. With BN, the binary case is very compelling, achieving within 1%p degradation compared to 8-bit's performance. The result for SBN is quite surprising, as it suggests that binary activation can be much better than few-bit activation; few-bit activation requires costly BN hardware whereas 1-bit activation works well even without BN. This is understandable if we remember that the BN problem is about weight precision, which is fixed to 1-bit in our case; thus, increasing weight precision doesn't help.

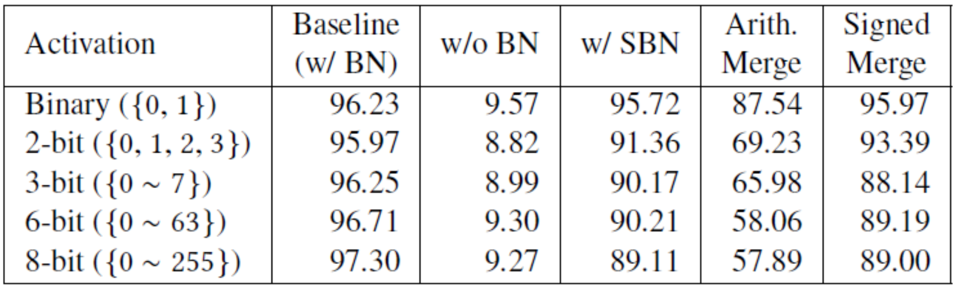
The 5th column of Table 4 lists the test accuracy after merging BN layers using (4) and (5) (referred to as arithmetic merge). As predicted, large errors are introduced by the arithmetic merge. Note that the merged version is mathematically the same as the baseline if weights were not quantized. Thus this accuracy drop is due to the forced quantization of w′ to 1-bit (see also Section 4.5). On the other hand, if we use (6) and (7) (referred to as signed merge), the accuracy is preserved in the case of binary activation.2 Since multi-bit activation could be thought of as a "superset" of the binary case, we apply the signed merge to multi-bit activation as well. However, the result shows that the binary case is clearly the best. This is because multi-bit activation uses a different activation function (tanh), and signed merge is mathematically correct only if it is followed by the sign-activation layer.

While the above results are without retraining, retraining didn't improve the performance at all. We also tried training the networks from scratch, starting from the merged version, which didn't train at all, probably due to the absence of BN layers.

In summary the results in this section suggest the following. First, BN is essential for binary-weight networks. Second, hardware-friendly BN implementations such as SBN can recover the accuracy to some degree. Third, the best option, at least for the MNIST network, is to use binary activation if weights are binary, meaning that for our application multi-bit analog activation is not necessary.

In addition, these results demonstrate that we must consider network's inference accuracy using actual training experiments in order to determine the best hardware architecture satisfying both efficiency and quality of result requirement, reinforcing the need for techniques such as ours.

Table 4 Activation precision and BN exploration (MNIST)



Activation	Baseline (w/ BN)	w/o BN	w/ SBN	Arith. Merge	Signed Merge
Binary ({0, 1})	96.23	9.57	95.72	87.54	95.97
2-bit ({0, 1, 2, 3})	95.97	8.82	91.36	69.23	93.39
3-bit ({0 ∼ 7})	96.25	8.99	90.17	65.98	88.14
6-bit ({0 ∼ 63})	96.71	9.30	90.21	58.06	89.19
8-bit ({0 ∼ 255})	97.30	9.27	89.11	57.89	89.00

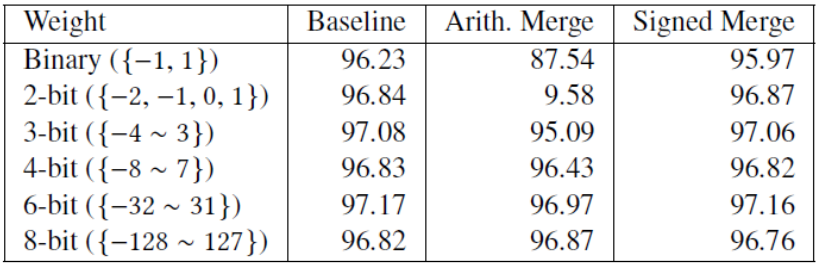
4.5 Additional Experiment on BN Merging

To see if the accuracy drop in BN merging is really caused by binary weight, we perform an additional experiment. Table 5 shows the result of arithmetic vs. signed merge applied to the MNIST DNN with multi-bit weights. This result is slightly different from that of Table 3, since the size of the network is fixed here. Nonetheless, the result shows that the network with merged BN can achieve near-baseline accuracy if weight precision is 3-bit or higher, confirming that accuracy drop in BN merging is due to low-precision weights.

Table 6 summarizes our findings regarding BN merging (see Tables 4 and 5). While signed merge was proposed before for binarized neural networks [26], it has not been applied to multi-bit networks. We not only extend signed merging for multi-bit networks, but also propose SBN, which can be the best option in certain cases.

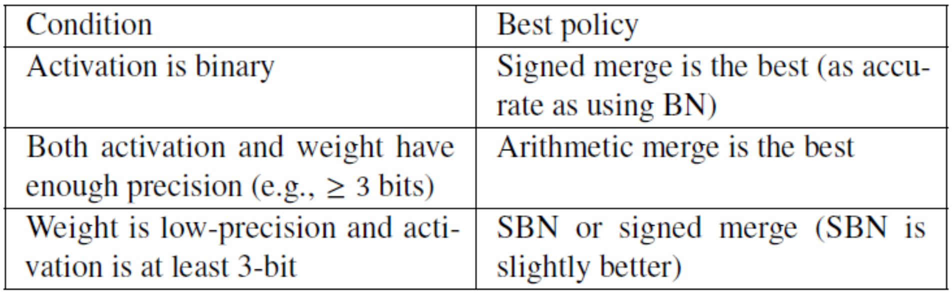
Additionally, this result shows that low-precision analog activation may not be very advantageous; rather, binary activation multi-bit weight can be a much better combination due to signed merge.

Table 5. MNIST multi-bit weight accuracy(%) after merging BN (activation: 1-bit, 400-128-128-10)



Weight	Baseline	Arith. Merge	Signed Merge
Binary ($\{-1, 1\}$)	96.23	87.54	95.97
2-bit ($\{-2, -1, 0, 1\}$)	96.84	9.58	96.87
3-bit ($\{-4 \sim 3\}$)	97.08	95.09	97.06
4-bit ($\{-8 \sim 7\}$)	96.83	96.43	96.82
6-bit ($\{-32 \sim 31\}$)	97.17	96.97	97.16
8-bit ($\{-128 \sim 127\}$)	96.82	96.87	96.76

Table 6. Best BN implementation policy for analog computing



Condition	Best policy
Activation is binary	Signed merge is the best (as accurate as using BN)
Both activation and weight have enough precision (e.g., $\geq$ 3 bits)	Arithmetic merge is the best
Weight is low-precision and activation is at least 3-bit	SBN or signed merge (SBN is slightly better)

Chapter V

HARDWARE EFFICIENCY EVALUATION

5.1 Architecture Detail

Figure 1 illustrates the top-level architecture of our optimized design. The network is a binarized neural network, with each layer having 400-128-128-10 neurons. Though our design is small, it is agreeable because we are aiming at a chip that can be implemented on a very small silicon area with an existing ReRAM fabrication backend, which is not necessarily state-of-the-art. The total number of ReRAM crossbar arrays is 6, each being no greater than 128×128. The amount of peripheral circuit is minimal as it uses no DAC/ADC except for the primary output.

Each layer consists of one or more ReRAM crossbar arrays surrounded by peripheral circuit. To represent negative weight, we use a reference current of value (ILRS + IHRS)/2 as illustrated in Figure 2a. This scheme, which is simpler than using two arrays, is possible due to our weights being binary. In our implementation, we create the reference current using a fixed resistor, which is more reliable than programming a ReRAM with half the HRS.

Each column of a crossbar array has two outputs: (i) ISUM, which is the result of a dot-product between the input and a weight vector, and (ii) the reference current (summed along the column, thus having the value of (ILRS +IHRS)/2 ·N, where N is the number of rows). The reference current represents how much ISUM is shifted. Thus the I-V converter uses the reference current as a threshold to determine the output (see Figure 2b). The I-V converter also serves as the activation function, which is a simple binary function. Current mirror is used before the I-V converter to minimize the output load effect.

On the input side, we have flip-flips (for optional pipelining), as well as drivers and buffers for programming ReRAMs. Finally, the output of the last layer is encoded as a binary number to be output through I/O pins.

INPUT (WL)	WEIGHT (RRAM)	OUTPUT	
		(I_{SUM})	(V_{ACT})
0	-1 (HRS)	0 $((I_{LRS}+I_{HRS})/2)$	0
0	+1 (LRS)	0 $((I_{LRS}+I_{HRS})/2)$	0
1	-1 (HRS)	-1 (I_{HRS})	-1 (VSS)
1	+1 (LRS)	+1 (I_{LRS})	+1 (VDD)

(a)

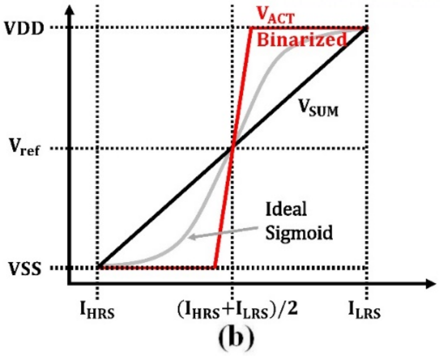


Figure 2. (a) Operating table and (b) I-V converter’s I/O curve

5.2 Details of Circuits

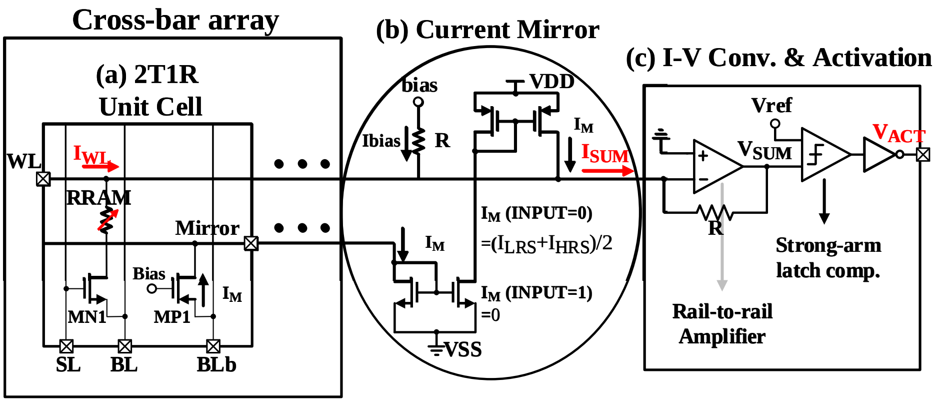


Figure 3. (a) 2T1R Unit Cell, (b) Current Mirror and (c) I-V Converter & Activation

Figure 3 shows the details of the circuits from the cell to activation function. Referring (a) of figure 2, when input is 0, the current of (ILRS +IHRS)/2 is generated from MP1 while MN1 is off. The half current is mirrored and goes to rail-to-rail amplifier which converts the current into the voltage such as Vsum. Then, the strong-arm-latch comparator analyze which is larger between Vsum and Vref. For the case of input is 1, MN1 is on and MP1 is off. The current which is determined to the states of RRAM, go to the rail-to-rail directly.

5.3 Comparison with Previous Work

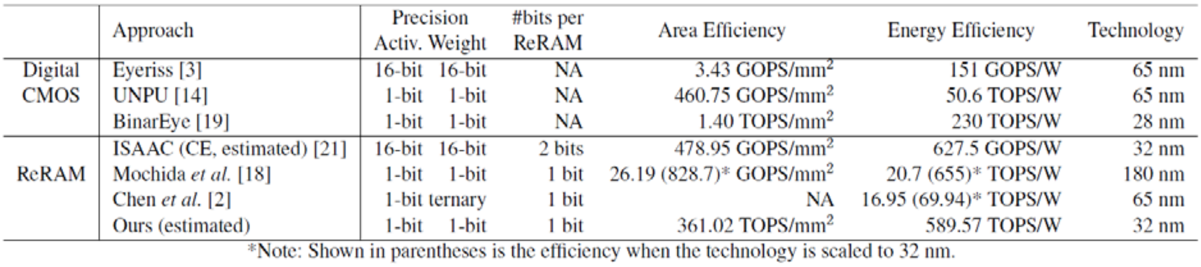
To evaluate the efficiency of our architecture, we follow the same modeling framework as in [21], except that the new components such as current mirror and I-V converter are implemented using Cadence Virtuoso up to layout, from which we obtain area and power. Figure 4 shows the area and power breakdown of our design. The operating frequency is determined to be 100 MHz from bandwidth analysis of I-V converter circuit.

Table 7 summarizes the comparison. Note that the numbers, taken from the literature, are based on different process technologies. Clearly, ReRAM-based hardware gives much better area and energy efficiency than digital CMOS-based neural networks, whereas the latter has more flexibility and scalability for larger neural networks. Among the ReRAM-based ones, we observe significant difference in hardware efficiency. To facilitate comparison we also provide the technology-scaled efficiency numbers; however, these numbers should be taken with caution as analog components may not be as easily scaled as digital components.

Nevertheless, the comparison shows that our design is much more efficient than ISAAC, whose area and power numbers are also estimated using the same methodology. There are several factors for this difference. First, ISAAC is comprised of more than 2000 blocks, each of which is about the size of our design. Consequently, ISAAC uses a large portion of area to non-computing components (e.g., eDRAM buffers and routers). If we consider only one such block (called IMA in [21]), its area efficiency is increased to 1.55 TOPS/mm2. Second, ISAAC implements 16-bit multiplication using 2-bit ReRAMs and assumes much higher operating frequency (1.2 GHz vs 100 MHz of ours). Converted into 1-bit multiplication on 1-bit ReRAMs, the effective efficiency is increased by 16 × 8-fold (at 1.2 GHz). Finally, the remaining gap is explained by the fact that crossbar arrays in ISAAC-CE generate 8 outputs per cycle due to the ADC bottleneck. By contrast ours generates all outputs at the same time, which can give up to 16× speedup for 128-column array. To sum, the main advantage of our design comes from precision optimization and ReRAM array-level parallelism.

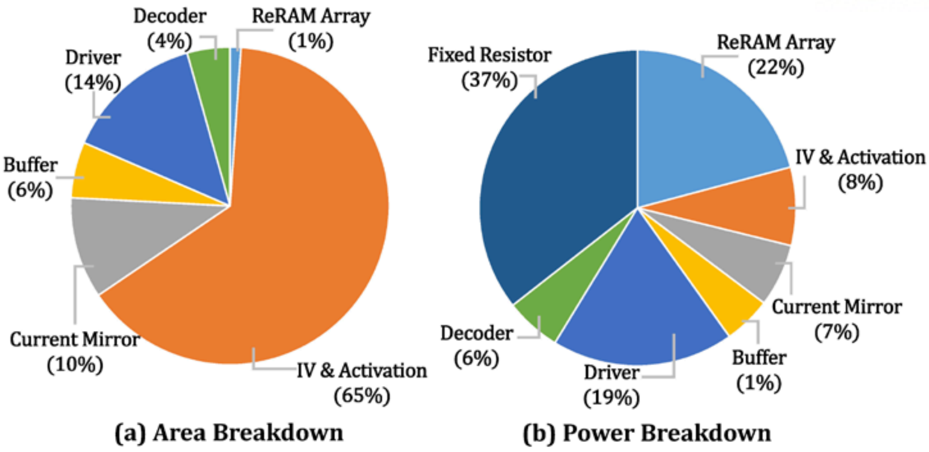
In terms of energy efficiency, ours is similar to that of [18], which is also custom-designed for MNIST with 1-bit precision. The other work [2], which also targets MNIST only, has much lower energy efficiency due to on-chip learning. All in all, our approach can generate very competitive design in terms of area and energy efficiency while ensuring high accuracy.

Table 7. Comparison with previous work(efficiency is peak efficiency)



	Approach	Precision Activ.	 Weight	#bits per ReRAM	Area Efficiency	Energy Efficiency	Technology
Digital CMOS	Eyeriss [3]	16-bit	16-bit	NA	3.43 GOPS/mm^2	151 GOPS/W	65 nm
	UNPU [14]	1-bit	1-bit	NA	460.75 GOPS/mm^2	50.6 TOPS/W	65 nm
	BinarEye [19]	1-bit	1-bit	NA	1.40 TOPS/mm^2	230 TOPS/W	28 nm
ReRAM	ISAAC (CE, estimated) [21]	16-bit	16-bit	2 bits	478.95 GOPS/mm^2	627.5 GOPS/W	32 nm
	Mochida *et al.* [18]	1-bit	1-bit	1 bit	26.19 (828.7)* GOPS/mm^2	20.7 (655)* TOPS/W	180 nm
	Chen *et al.* [2]	1-bit	ternary	1 bit	NA	16.95 (69.94)* TOPS/W	65 nm
	Ours (estimated)	1-bit	1-bit	1 bit	361.02 TOPS/mm^2	589.57 TOPS/W	32 nm

*Note: Shown in parentheses is the efficiency when the technology is scaled to 32 nm.



(a) Area Breakdown

(b) Power Breakdown

Figure 4. Area and power breakdown

Chapter VI

CONCLUSION

In this paper we presented an extremely cost-efficient neural network based on ReRAM crossbar array and analog computing. By replacing costly peripheral circuitry such as ADC/DAC with analog counterparts, we can achieve very high efficiency per area and energy, while ensuring that our scaled-down neural network processor does give high quality of result. One of the key ingredients in such low-precision networks is batch normalization, for which we examined multiple methods. Our design framework can be useful for applications where precision (or quality of result) can be traded off for higher hardware efficiency such as in IoT devices.

REFERENCE

[1] E. R. Berikaa et al. 2018. Multi-Bit RRAM Transient Modelling and Analysis. In 2018 30th International Conference on Microelectronics (ICM). 232–235.

[2] Wei-Hao Chen et al. 2019. CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. Nature Electronics 2, 9 (2019), 420–428.

[3] Y. Chen et al. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. IEEE Journal of Solid-State Circuits 52, 1 (Jan 2017), 127–138.

[4] Zhiyong Cheng et al. 2015. Training Binary Multilayer Neural Networks for Image Classification using Expectation Backpropagation. CoRR abs/1503.03562 (2015). arXiv:1503.03562

[5] P. Chi et al. 2016. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). 27–39.

[6] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. CoRR abs/1602.02830 (2016). arXiv:1602.02830

[7] E. Giacomin et al. 2019. A Robust Digital RRAM-Based Convolutional Block for Low-Power Image Processing and Learning Applications. IEEE Transactions on Circuits and Systems I: Regular Papers 66, 2 (Feb 2019), 643–654.

[8] Zhezhi He et al. 2019. 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 2019. ACM, 57.

[9] Itay Hubara et al. 2017. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. J. Mach. Learn. Res. 18, 1 (Jan. 2017), 6869–6898.

[10] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. (Feb. 2015).

[11] Alex Krizhevsky. 2012. Learning Multiple Layers of Features from Tiny Images. University of Toronto (May 2012).

[12] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25. Curran Associates, Inc., 1097–1105.

[13] Y. Lecun et al. 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 11 (Nov 1998), 2278–2324.

[14] J. Lee et al. 2019. UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision. IEEE Journal of Solid-State Circuits 54, 1 (Jan 2019), 173–185.

[15] Can Li et al. 2018. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. Nature Communications 9, 1 (2018), 2385.

[16] Fengfu Li, Bo Zhang and Bin Liu. 2016. Ternary Weight Networks. CoRR abs/1605.04711 (2016). arXiv:1605.04711

[17] S. Lim, M. Kwak and H. Hwang. 2018. Improved Synaptic Behavior of CBRAM Using Internal Voltage Divider for Neuromorphic Systems. IEEE Transactions on Electron Devices 65, 9 (Sep. 2018), 3976–3981.

[18] R. Mochida et al. 2018. A 4M Synapses integrated Analog ReRAM based 66.5 TOPS/W Neural-Network Processor with Cell Current Controlled Writing and Flexible Network Architecture. In 2018 IEEE Symposium on VLSI Technology. 175–176.

[19] Bert Moons et al. 2018. BinarEye: An Always-On Energy-Accuracy-Scalable Binary CNN Processor With All Memory On Chip in 28nm CMOS. CoRR abs/1804.05554 (2018). arXiv:1804.05554

[20] Olga Russakovsky et al. 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115, 3 (2015), 211–252.

[21] A. Shafiee et al. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). 14–26.

[22] X. Sun et al. 2018. Fully parallel RRAM synaptic array for implementing binary neural network with (+1, 1) weights and (+1, 0) neurons. In 2018 23rd Asia and South Pacific Design Automation Conference. 574–579.

[23] Tianqi Tang et al. 2017. Binary convolutional neural network on RRAM. In 2017 22nd Asia and South Pacific Design Automation Conference. 782–787.

[24] Xiaowei Xu et al. 2018. Scaling for edge inference of deep neural networks. Nature Electronics 1, 4 (2018), 216–222.

[25] C. Xue et al. 2019. 24.1 A 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6ns Parallel MAC Computing Time for CNN Based AI Edge Processors. In 2019 IEEE International Solid- State Circuits Conference - (ISSCC). 388–390.

[26] H. Yonekawa and H. Nakahara. 2017. On-Chip Memory Based Binarized Convolutional Deep Neural Network Applying Batch Normalization Free Technique on an FPGA. In 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). 98–105.

[27] Kyungjean Yoon et al. 2016. Comprehensive Writing Margin Analysis and its Application to Stacked one Diode-One Memory Device for High-Density Crossbar Resistance Switching Random Access Memory. Advanced Electronic Materials 2 (Sep. 2016).