Efficient Hardware Implementation of Cellular Neural Networks with Incremental Quantization and Early Exit

XIAOWEI XU, QING LU, and TIANCHEN WANG, University of Notre Dame, USA
YU HU, Huazhong University of Science and Technology, China
CHEN ZHUO, Zhejiang University, China
JINGLAN LIU and YIYU SHI, University of Notre Dame, USA

Cellular neural networks (CeNNs) have been widely adopted in image processing tasks. Recently, various hardware implementations of CeNNs have emerged in the literature, with Field Programmable Gate Array (FPGA) being one of the most popular choices due to its high flexibility and low time-to-market. However, CeNNs typically involve extensive computations in a recursive manner. As an example, to simply process an image of 1,920 × 1,080 pixels requires 4–8 Giga floating point multiplications (for 3 × 3 templates and 50–100 iterations), which needs to be done in a timely manner for real-time applications. To address this issue, in this article, we propose a compressed CeNN framework for efficient FPGA implementations. It involves various techniques, such as incremental quantization and early exit, which significantly reduces computation demands while maintaining an acceptable performance. Particularly, incremental quantization quantizes the numbers in CeNN templates to powers of two, so that complex and expensive multiplications can be converted to simple and cheap shift operations, which only require a minimum number of registers and logical elements (LEs). While a similar concept has been explored in hardware implementations of Convolutional Neural Networks (CNNs), CeNNs have completely different computation patterns, which require different quantization and implementation strategies. Experimental results on FPGAs show that incremental quantization and early exit can achieve a speedup of up to 7.8× and 8.3×, respectively, compared with the state-of-the-art implementations, while with almost no performance loss with four widely adopted applications. We also discover that different from CNNs, the optimal quantization strategies of CeNNs depend heavily on the applications. We hope that our work can serve as a pioneer in the hardware optimization of CeNNs.

CCS Concepts: • **Hardware** → **Hardware accelerators**; • **Computer systems organization** → *Neural networks*; *Embedded hardware*;

Additional Key Words and Phrases: Cellular neural networks, quantization, acceleration, FPGA

ACM Reference format:
Xiaowei Xu, Qing Lu, Tianchen Wang, Yu Hu, Chen Zhuo, Jinglan Liu, and Yiyu Shi. 2018. Efficient Hardware Implementation of Cellular Neural Networks with Incremental Quantization and Early Exit. *J. Emerg. Technol. Comput. Syst.* 14, 4, Article 48 (November 2018), 20 pages.
https://doi.org/10.1145/3264817

Authors' addresses: X. Xu, Q. Lu, and T. Wang, 356C, Fitzpatrick Hall, University of Notre Dame, Computer Science and Engineering, South Bend, IN, 46655, USA; emails: {xxu8, qlu2, twang9}@nd.edu; Y. Hu, Room 502, Enming Building, Huazhong University of Science and Technology, Wuhan, China; email: bryanhu@hust.edu.cn; C. Zhuo, Room 501, Laoshengyi Building, Zhejiang University, Hangzhou, China; email: czhuo@zju.edu.cn; J. Liu and Y. Shi, Room 356C, Fitzpatrick Hall, University of Notre Dame, Computer Science and Engineering, South Bend, IN, 46655, USA; emails: {jinglan.liu.194, yshi4}@nd.edu.

1550-4832/2018/11-ART48 $15.00
https://doi.org/10.1145/3264817

1 INTRODUCTION

Cellular Neural Networks (CeNNs) can model the working principles of many sensory parts of human brains. Different from Convolutional Neural Networks (CNNs) [31, 36], which are most powerful in classification related tasks, CeNNs are generally good at various image processing areas such as noise cancellation [15], edge detection [6], path planning [9], and segmentation [5]. Due to the complex nature of these tasks and the associated real-time requirements in many applications, hardware implementations of CeNNs have remained an active research topic in the literature.

The structure of CeNNs makes them a natural fit for analog implementations. Many studies exist along this direction [1, 8, 17, 23]. The advantages of analog implementations include high performance with an extremely fast convergence rate [32, 33, 37] and the convenience of integrating them into image sensors for direct processing of captured data. However, these analog implementations suffer from Input/Output (I/O) and data precision problems. First, they require that each input corresponds to a unique neuron cell, resulting in too many I/O ports. For example, recent implementation [1] can only support 256×256 pixels at its most, which is far from the processing requirement of mainstream images, e.g., $1{,}920 \times 1{,}080$ pixels. Second, analog circuits are prone to noise, which limit the output precision to 7 bits or below [38]. As a result, analog implementation cannot even process regular 8-bit gray images.

In view of the above issues, digital implementations of CeNNs have been proposed, where data is quantized with approximation. Tens to hundreds of iterations are needed in the discretized process and as a result, the computational complexity of digital CeNNs is very high. For example, to process an image of $1{,}920 \times 1{,}080$ pixels requires 4–8 Giga operations (for 3×3 templates and 50–100 iterations), which needs to be done in 40ms or below for real-time video streaming.

To tackle the computation challenge, CeNN accelerations on digital platforms such as ASICs [14, 16], GPUs [21], and FPGAs [2, 18–20, 38, 39] have been explored, with FPGA among the most popular choices due to its high flexibility and low time-to-market. The work [2] presented a baseline design with several applications, while the study in Reference [20] took advantage of reconfigurable computing for CeNNs. Recently, the CeNN implementation for binary images was demonstrated [19]. Expandable and pipelined implementations were achieved on multiple FPGAs [18]. Taking advantage of the structure in Reference [18], the work in Reference [38] implemented a high throughput real-time video streams system, which is further improved to be a complete system for video processing [39]. All the three works share the same architecture for CeNN computation. Due to the large number of multiplications needed in CeNNs, the limited quantity of embedded multipliers in an FPGA become the bottleneck for further improvement. For example, in Reference [18], 95%–100% of the embedded multipliers are used. However, it is interesting to note that the utilization rates of LEs and registers are only 5% and 2%, respectively, which is natural to expect as not many logic operations are needed. However, in a mainstream FPGA, LEs and registers count for significantly larger portion of the total programmable resources than embedded multipliers. For example, LEs and registers occupy 95.4% of the core area while embedded multipliers only 4.6% for a EP3LS340 FPGA [29]. Such an unbalanced resource utilization apparently cannot attain the best possible speed of the CeNN being implemented, and an improved strategy is strongly desired.

A naive approach for potential improvement is to use LEs and registers to implement additional multipliers. This technique, although straightforward, is very inefficient due to the high cost associated. For example, it takes 676 LEs and 486 shift registers to implement an 18-bit multiplier. For an XC4LX25 FPGA, all the LEs and registers can only contribute 42% additional multipliers. Apparently, such an approach will not lead to significant improvement, and we try to address the

problem through an alternative approach, i.e., by completely eliminating the need of multipliers. From basic boolean algebra, we know that the multiplication of any number with powers of two can simply be done with logic shift, which only requires a small number of LEs and registers to achieve. Inspired by this, we can quantize the numbers in CeNN templates to powers of two, so that we can make full use of the abundant LEs and registers in FPGAs. An extra benefit from this approach is that LEs and registers are much more flexible for placement and routing, leading to higher clock frequencies. While this can lead to significantly higher resource utilization rate and reduced computational complexity, many interesting questions still remain. For example, how would such quantizations affect the final CeNN accuracy? What is the impact of different quantization strategies? Note that quantization to powers of two has been explored in the context of CNNs [40], but as detailed in Section 2.3, the difference in computation structures between CeNNs and CNNs warrants a separate investigation for CeNNs. And indeed, we figure out that the answers to these questions are different for the two.

Another feature of CeNN is that the output of each neuron changes gradually over time (or iteration times for discrete approximation). It should be noted that though the computation is performed in analog circuit for CeNN, its corresponding discrete approximation performed on FPGAs has the same shape. Thus, for specific applications, we can complete CeNN computations earlier than general computations with an acceptable performance.

In this article, we present a compressed CeNN framework for computation reduction in CeNNs [34, 35]. Particularly, we systematically put forward two optimizations: incremental quantization and early exit. Incremental quantization contains iterative procedures including parameter partition, parameter quantization, and re-training. We propose five different strategies, including random strategy, pruning inspired strategy, weighted pruning inspired strategy, nearest-neighbor strategy, and weighted nearest-neighbor strategy for incremental quantization. Out of the five only pruning-inspired strategy and random strategy have been adopted in incremental quantization of CNNs [40] due to the differences in their computation patterns. Early exit can fulfill the computation earlier than general computation with almost no accuracy loss. We have conducted extensive experiments with four widely used applications to evaluate the proposed framework. We then implement these quantized CeNNs on FPGAs with multiplications realized by shift operations. Based on CeNN template structures, sparsity-induced and repetition-induced optimizations for quantized templates are also exploited for situations where resources are extremely limited. Experimental results on FPGAs show that incremental quantization and early exit can achieve a speedup of up to 7.8× and 8.3×, respectively, compared with the state-of-the-art implementations, while with almost no performance loss with four widely-adopted applications.

The remainder of the article is organized as follows. Section 2 introduces backgrounds and motivation of the article. The proposed compressed CeNN framework and the optimized hardware implementation are presented in Section 3. Experiments and discussion are provided in Section 4 and concluding remarks are given in Section 5.

2 PRELIMINARIES

In this section, we first introduce the mathematical details of CeNNs, followed by template learning algorithms, which are used to train CeNNs. Particularly, a widely used template learning algorithm, Particle Swarm Optimization (PSO)[13], is presented in detail and adopted in this article.

2.1 Cellular Neural Networks

Different from the prevalent CNNs superior for classification tasks, CeNN model is inspired by the functionality of visual neurons, and a mass of neuron cells is connected with neighboring ones. Only adjacent cells can interact directly with each other. This is a significant advantage for

hardware implementation, resulting in much less routing complexity and area overhead. CeNNs are superior at image processing tasks that involves sensory functions, such as noise cancellation, edge detection, path planning, segmentation, and so on. For the widely used 2D CeNN with space-invariant templates, the dynamics of each cell state with an M×N rectangular cell array [3] are as follows:

$$\dot{x}_{i,j}(t) = -x_{i,j}(t) + \sum_{k,l=-N}^{N} (A_{k,l}(t)y_{i+k,j+l}(t) + B_{k,l}(t)u_{i+k,j+l}(t)) + I(t), \tag{1}$$

$$y_{i,j}(t) = f(x_{i,j}(t)) = 0.5 \times (|x_{i,j}(t) + 1| - |x_{i,j}(t) - 1|), \tag{2}$$

where $1 \le i \le M$, $1 \le j \le N$, $A_{k,l}(t)$ is the feedback coefficient template, $B_{k,l}(t)$ is the input coefficient template, $I(t)$ is the bias, and $x_{i,j}(t)$, $y_{i+k,j+l}(t)$ and $u_{i+k,j+l}(t)$ are the state, output and input of the cell, respectively. Note that $A_{k,l}(t)$, $B_{k,l}(t)$ and $I(t)$ are time-variant templates, and t can be removed when time-invariant templates are used. For efficient implementation on a digital platform (e.g., CPU, GPU, FPGA), discrete approximation of CeNN is obtained by applying Euler approximation as shown in Equations (3), (4), and (5):

$$x_{i,j}(t) \cong (x_{i,j}(n+1) - x_{i,j}(n))/\Delta t, \tag{3}$$

$$x_{i,j}(n+1) = x_{i,j}(n) + \Delta t(-x_{i,j}(n) + I(n) + \sum_{k,l=-N}^{N} (A_{k,l}(n)y_{i+k,j+l}(n) + B_{k,l}(n)u_{i+k,j+l}(n))), \tag{4}$$

$$y_{i,j}(n) = f(x_{i,j}(n)) = 0.5 \times (|x_{i,j}(n) + 1| - |x_{i,j}(n) - 1|). \tag{5}$$

Delayed CeNN is a special type of CeNN described by adding $\sum_{k,l=-N}^{N}(D_{i,j}(n)g(x_{k,l}(n), y_{k,l}(n), u_{k,l}(n))$ to Equation (4), where g is usually a piece-wise constant function. Delayed CeNN will also be considered in this article when the effectiveness of incremental quantization is discussed. Please refer to Reference [3] for details. For the mainstream image size with 1,920 × 1,080 pixels, the total complexity is $1{,}920 \times 1{,}080 \times 39 \times 100 = 8.1 \times 10^9$ operations with 100 iterations (19 multiplications and 20 additions in each iteration). This warrants algorithms to speedup the computations.

2.2 Template Learning Algorithm and PSO Algorithm

Template learning is a widely studied and applied method to find satisfactory templates for CeNN-based applications, in which Genetic Algorithm (GA) [28] and Particle Swarm Optimization (PSO) [13] are two representatives. PSO is adopted in this article, while GA and other template learning method are also compatible with the framework to be proposed.

PSO finds solutions in a heuristic way by searching the solution space with multiple particles (swarm of potential solutions). In each iteration, PSO performs position update and object function calculation. Inspired by the social behavior of animals, the position update of each particle is affected by its past best position and the position of the current global best position as depicted by Equation (6), where $1 \le i \le N$, $1 \le d \le D$, N is the size of particles, D is the dimension of each particle, c_1 and c_2 are the acceleration coefficients, and r_1 and r_2 are random numbers with uniform distribution. $p_i(n+1)$ and $p_i(n)$ are the positions of the ith particle in iteration n and $n+1$, respectively. pb_n is the best position that the ith particle ever searches, and gb is the current best position among all particles. Inertia weight w controls the balance of the search algorithm between exploration and exploitation. A bound of $[min_d, max_d]$ is introduced for $p_{i,d}$ to limit the solution space.

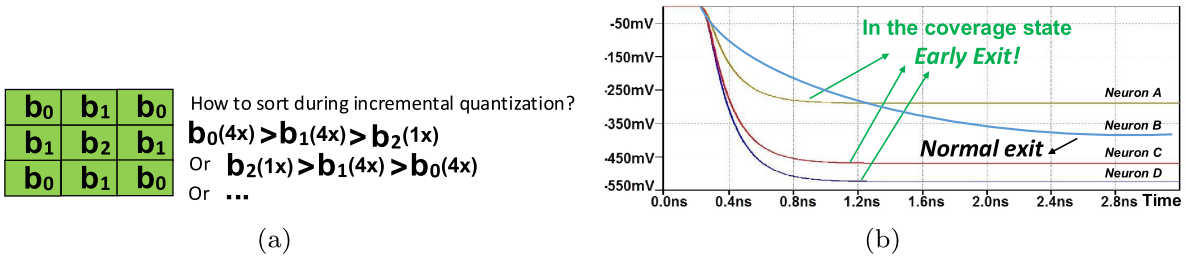


Fig. 1. Motivation illustration: (a) CeNN template for binary image noise cancellation application, (b) early exit in CeNN computations.

The object function for particles taking positions as input is designed according to applications:

$$p_{i,d}(n+1) = p_{i,d}(n) + \{w \times v_{i,d}(n) + c_1 r_1 \times (pb_{i,d} - p_{i,d}(n)) + c_2 r_2 \times (gb_d - p_{i,d}(n))\}. \quad (6)$$

2.3 Motivation

While hardware-oriented memory/computation compression and optimization of CNNs have been extensively studied recently [4, 11, 22, 25, 30, 40], little has been explored for CeNNs where memory consumption is not a problem and the focus is only on computational complexity.

The main difference between CeNNs and CNNs is that in CeNNs the parameters are coupled. The weight values in a CNN tend to be all unique. However, in CeNNs some parameters share the same values. For example, in Figure 1(a), a CeNN template (template B) for binary image noise cancellation [15] is shown. Only three different values exist for the nine parameters. As such, in Reference [40] the weights of CNNs are incrementally quantized in an order simply based on their magnitudes (pruning-inspired strategy). The same strategy may not work well for CeNNs, as a parameter with small magnitude may repeat multiple times thus playing a more important role than a parameter with a large magnitude but appearing only once. Furthermore, the training process of CNNs is mathematically optimal, while that of CeNNs is heuristic. This will also influence the performance of quantization strategies. Finally, the sparsity and repetition existing in CeNN templates provide some additional opportunity for further improvement when implemented in hardware.

Another difference that should be noted is that the output of each neuron changes gradually over time (or iteration times for discrete approximation). As shown in Figure 1(b), the value of four selected neurons varies with time. It should be noted that though the computation is performed in analog fashion for CeNN, its corresponding discrete approximation performed on FPGAs has the same shape. We can observe that there are many neurons (neuron A, C, and D) with a relatively shorter convergence time than others (Neuron B), which means early exit can be performed as the later values will not change over time. Thus, during a CeNN system evolution, significant amount of computation can be eliminated to save energy without sacrificing accuracy.

3 COMPRESSED CENN FRAMEWORK AND HARDWARE IMPLEMENTATION

In this section, we present the compressed CeNN framework followed by the details of the hardware implementation.

3.1 Incremental Quantization

Incremental quantization is an iterative process as shown in Figure 2. Each iteration completes three tasks: parameter partition, parameter quantization, and incremental re-training. We assume

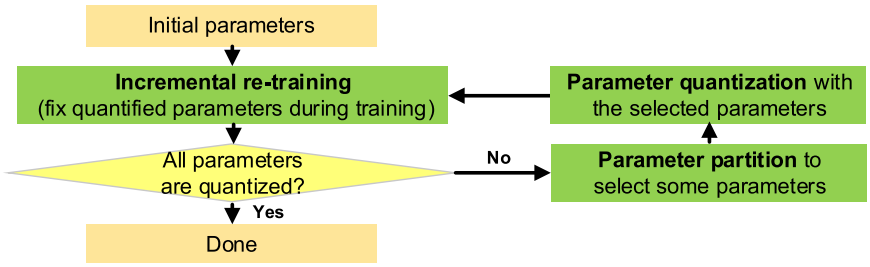


Fig. 2. The flowchart of incremental quantization.

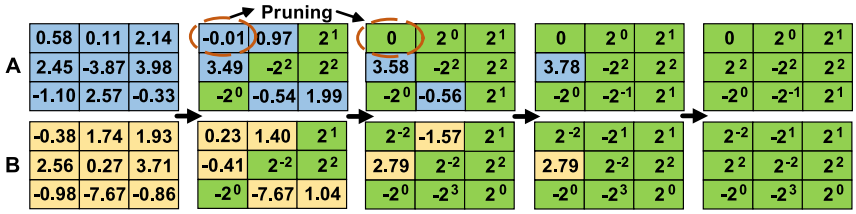


Fig. 3. An example of incremental quantization: in each iteration, parameter partition, parameter quantization, and incremental re-training are performed sequentially. Green cells represent quantized parameters.

that as a starting point, we have all parameters in the original templates before quantization well trained. An illustrative example of the process is shown in Figure 3 to facilitate understanding.

3.1.1 Parameter Partition. This task selects a subset of parameters not yet quantized (un-quantized parameters) to perform quantization. Two knobs exist in this task: parameter priority and batch size.

For the first knob, the pruning-inspired (PI) strategy has been well explored in quantization of CNNs [40], based on the consideration that weights with larger magnitudes contribute more to the result and thus should be quantized first. However, the parameters in CeNNs have some unique characteristics, which have been discussed in Section 2.3. To tackle the problem, we propose a nearest-neighbor (NN) strategy and a weighting method for the first knob. The combined weighted nearest-neighbor algorithm takes the number that a parameter appears in the template, defined as its repetition quantity (rq) as the reciprocal of the weight, and uses the difference between the parameter and its nearest power-of-two as distance to perform a weighted NN algorithm (WNN). The detail explanation of WNN algorithm is shown in Algorithm 1. Other combinations such as weighted pruning-inspired (WPI) strategy adopt the same weighting method but with PI to form WPI. A total of five strategies PI, WPI, NN (WNN with all weights set to 1), WNN and a random strategy (RAN) are compared in the experimental section.

For the second knob, batch size is the number of parameters selected in each iteration, which will affect re-training speed and quality. We propose to use two batch sizes, constant and log-scale. The former selects the same number of parameters in each iteration, while the latter picks a fixed percentage from the remaining un-quantized parameters, rounded to the nearest integer. Compared with constant batch size, log-scale batch size quantizes more parameters in the first several iterations and fewer toward the end.

3.1.2 Parameter Quantization. Before parameter quantization, the bit width should be defined first according to applications. Note that there are millions of parameters for CNN, and short bit width is always appreciated considering memory and computational consumption. However, CeNN usually has tens to hundreds of parameters (time-variant templates have more parameters than time-invariant templates), and bit width has no significant impact on memory consumption.

ALGORITHM 1: Weighted nearest-neighbor strategy

Input: un-quantized parameters uq_i, repeat quantity, rq_i, selected quantity, N, $1 \le i \le n$, n, the number of un-quantized parameters
Output: the most important N parameters
$neighbor = log_2 |(uq)|$; // get the power of the absolute value of the un-quantized parameters
for $i = 1$ **to** n **do**
  $md = (2^{floor(neighbor(i))} + 2^{floor(neighbor(i)+1)})/2$;
  if $md > |(uq(i))|$ **then**
    $nnDist(i) = |(uq(i))| - 2^{floor(neighbor(i))}$;
  else
    $nnDist(i) = 2^{floor(neighbor(i))+1} - |(uq(i))|$;
  end if
end for
$wnnDist = nnDist/rq$;
sort $wnnDist$ in ascending order;
output the first N parameters;

In addition, with power-of-two conversion multiplications can be done with logic shifts, and bit width will also have little impact on computation complexity. The only impact it will have is on the resource utilization of multipliers.

Suppose the quantization set is designed as depicted in Equation (7), where k and m indicate the range of quantization. The corresponding bit width bw is calculated as shown in Equation (8), where the extra one bit is the sign bit:

$$qs = \{\pm(2^k, ., 2^p, ., 2^m), 0\}, \quad k \le p \le m, \quad p, k, m \in \mathbb{Z}, \tag{7}$$

$$bw = Ceiling[log_2(2 \times (m - k + 1) + 1)] + 1. \tag{8}$$

With the quantization set, a parameter $uq(i)$ is quantized as shown in Equation (9). When the absolute value of a parameter is smaller than 2^{-k-1}, it will become zero after quantization and get pruned. Lower bit width can prune more parameters, at the cost of accuracy loss:

$$uq(i) = \begin{cases} 2^p & \text{if } 3 \times 2^{p-2} \le |uq(i)| < 3 \times 2^{p-1}; \\ & \quad k \le p \le m; \\ 2^m & \text{if } |uq(i)| \ge 2^m; \\ 0 & \text{if } |uq(i)| < 2^{-k-1}. \end{cases} \tag{9}$$

3.1.3 Incremental Re-training Algorithm. Usually, re-training algorithm is an optimal problem as shown in Equation (10), where P is the set of all the parameters. In incremental re-training algorithm, the optimal problem is revised as shown in Equation (11), where U and Q are the sets of un-quantized and quantized parameters, respectively. a_i and b_i are the lower and upper bounds for both P_i and U_i, respectively. Note that $P = Q \cup U$, and $U \cap Q = \emptyset$. In each iteration, a subset of U will be quantized and added to Q:

$$f = min\ obj(P),\ s.t.\ P_i \in [a_i, b_i], 0 \le i \le |P|, \tag{10}$$

$$f = min\ obj(U, Q),\ s.t. U_i \in [a_i, b_i], 0 \le i \le |U|. \tag{11}$$

Q will be fixed during the re-training process and only U is used for space searching. After multiple iterations, all the required parameters are quantized. It should be noted that the bias $I(n)$ in

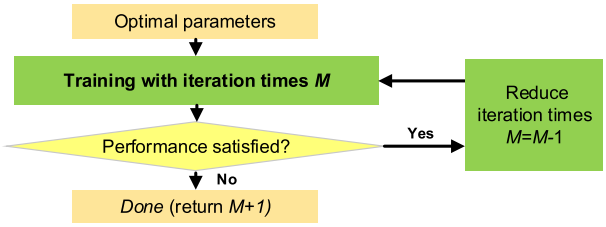


Fig. 4. Flowchart of early exit.

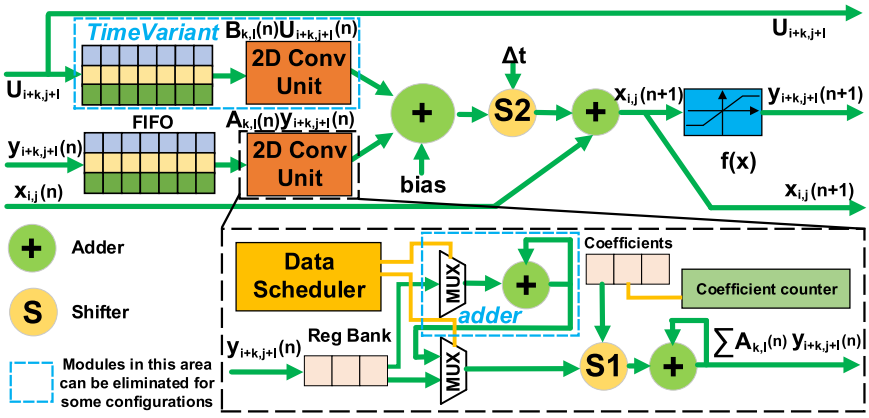


Fig. 5. Architecture of the optimized stage design.

Equation (4) for CeNN is not required to be quantized as it is not involved in multiplication. Therefore, another re-training iteration is required for the optimal bias when all the required parameters are quantized.

3.2 Early Exit

Early exit exploits the convergence character during the analog computation of CeNN: the output has a relatively larger variance in the early runtime (or first several iterations for digital approximation) and a much smaller variance in the afterward runtime. In this article, we adopt an experimental method to perform early exit as shown in Figure 4. Note that the performance is determined by specific applications. In this method, the possible iteration times will be explored from a large value to one, and the iteration time with an acceptable performance will be extracted.

3.3 Efficient Hardware Implementations

We base our work on the state-of-the-art FPGA CeNN implementations [18, 38, 39], which is expandable, highly parallel, and pipelined. The basic element of the architecture is the stage module, which handles all the processes in one iteration corresponding to Equation (4) for $1 \leq i \leq M$, $1 \leq j \leq N$. Multiple stages are connected sequentially for multiple iterations to form a layer, which processes the input in a pipelined manner. Furthermore, multiple layers can be connected sequentially for more complex processing or be distributed in parallel for a higher throughput. Note that First In First Out (FIFO) are used between adjacent stages to store the temporary results of each stage (or each iteration), and they are configured as single-input multiple-output memories. Please refer to FPGA implementations in References [18, 38] for more details.

Our efficient hardware implementation focuses on the optimization of the stage design as shown in Figure 5. Two optimizations are performed: multiplication simplification and data movement optimization. First, with incremental quantization, simplification can be achieved by replacing multiplications with shift operations. The detailed hardware implementation will be discussed in

Table 1. Comparison of Resource Utilization Between 18-bit Multipliers Implemented Using Shifter Modules of Various Configurations $S1(m)$ and $S2(m)$ (with Different m as Defined in Equation (7), $k = -m$ for $S1$, and k=0 for $S2$) and a Direct Implementation of an 18-bit Multiplier Using LEs and Registers

Module	$S1(0)$	$S1(1)$	$S1(2)$	$S1(3)$	$S1(4)$	$S1(5)$	$S2(7)$	*Multiplier*
LEs	39	44	50	80	109	105	80	676
Registers	39	42	45	47	50	52	75	486

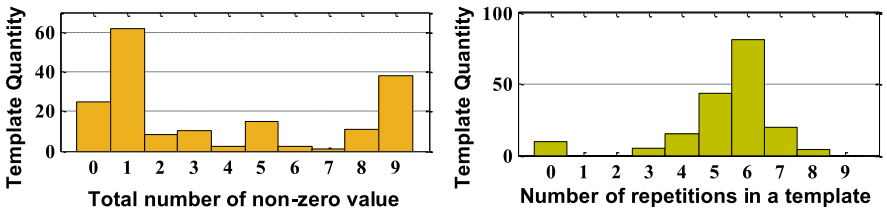


Fig. 6. Illustration of (a) sparsity and (b) repetition characteristic with 174 CeNN templates.

Section 3.3.1. Second, when FPGA resource is extremely limited (e.g., for low-end FPGAs), data movement optimization can be performed utilizing the sparsity and repetition in CeNN templates. As will be discussed later in Section 3.3.2, in many applications CeNN templates naturally involves zero or repeated parameters. With incremental quantization, more zeros are yielded leading to higher sparsity and the small quantization set introduces a larger number of repetitions. Data movement optimization can minimize the number of computations needed. The details will be discussed in Section 3.3.2.

The optimized stage can be configured for both time-invariant templates and time-variant templates. Note that the FPGA implementation [38] is dedicated to CeNN with time-invariant templates, while Reference [18] is for time-variant. The $TimeVariant$ part in Figure 5 is specific for time-variant templates, and can be eliminated in the configuration for time-invariant ones.

3.3.1 Shifter Module. In Figure 5, shifter $S1$ is for multiplications in CeNNs and $S2$ is for discrete approximation involved with Δt in Equation (4). Usually Δt is very small, and the hardware implementation of $S2$ in this article is designed to support $\Delta t = 2^s$, where $-7 \leq s \leq 0$, $s \in \mathbb{Z}$. Note that when Δt is configured to 2^0 or 1, the computation is transformed to discrete CeNN [7].

Table 1 provides an illustrative comparison of resource utilization between multipliers implemented using shifter modules of various configurations and a direct implementation of multiplier using LEs and registers. It can be noticed that the shifter module consumes much fewer resources than the general implementation, such that more multiplications can be placed on FPGAs for higher performance and speed. It should be pointed out that multiple shifters can be adopted in the 2D convolutional module.

3.3.2 Data Scheduler Module. Data scheduler module exploits the sparsity and repetition of parameters in CeNN templates. We analyzed 87 tasks from 79 applications [12], and in total, 174 templates are examined (each task has two templates: template A and template B). All the templates are 2D 3×3 each having nine parameters. The corresponding sparsity and repetition are shown in Figure 6(a). In Figure 6(a), we discover that a majority of templates have zero values, and more than half have only three or less non-zero parameters. Therefore, ignoring multiplications with zeros will give a significant improvement in efficiency.

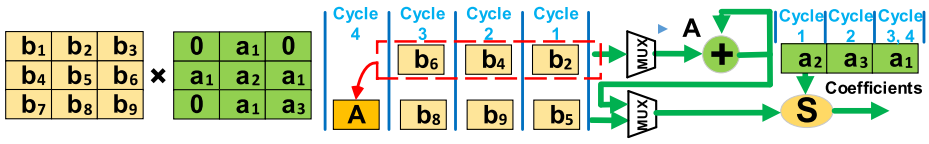


Fig. 7. Illustration of sparsity-induced and repetition-induced optimizations.

Figure 6(b) depicts the histogram of the parameter repetition in all the 174 templates. We can see that in most of the templates, about 5–6 parameters are repeated values. With repeated parameters, we can also take advantage of the associative law for repetition-induced optimization, e.g., $a_1 \times b_1 + a_1 \times b_2 + a_1 \times b_3 = (b_1 + b_2 + b_3) \times a_1$, and hence three multiplications are optimized to only one.

Note that these optimizations seem to be straightforward and automatic in software synthesis, but for hardware implementations detailed attention is needed. An illustration of optimization with sparsity and repetition is shown in Figure 7. With sparsity-induced optimization, we only take the non-zero parameters into consideration, and three multiplications can be eliminated. An adder (only consumes 10 LEs in the design) is utilized to calculate the sum A of b_2, b_4 and b_6 in parallel with the shifter module. The shifter module calculates $b_5 \times a_2$, $b_9 \times a_3$, and $b_8 \times a_1$ in the first three cycles, and computes $A \times a_1$ in the forth. Thus, totally it takes four cycles rather than nine cycles to calculate Equation (8). Specifically, sparsity-induced optimization reduces the computation time from nine cycles to six, and repetition-induced optimization reduces it from six to four.

The power of sparsity-induced and repetition-induced optimizations varies with different applications. Note that if the number of shifters adopted in the 2D convolution module is larger than one, repetition-induced optimization can be eliminated as it contributes much less compared with the shifters. If the number of shifters equals that of the coefficients, which is also the situation to achieve the highest throughput, then repetition-induced optimization can also be eliminated as all multiplications can be processed in only one cycle. Therefore, the two optimizations are only for situations with very limited resources.

4 EXPERIMENTS

In this section, we first evaluate the performance of incremental quantization and early exit discussed in Section 3. Then, we implement the quantized CeNNs on FPGAs and compare their speed with state-of-the-art works.

4.1 Performance Evaluation

We choose four applications, i.e., binary image noise cancellation, grey image noise cancellation, texture segmentation, and medical image segmentation. A total of 10 incremental quantization strategies are evaluated: five partition strategies (RAN, PI, WPI, NN (WNN with all weights set to 1), and WNN) in combination with two batch sizes (constant and log-scale). For compact presentation, we use postfix-C and -L to denote constant and log-scale batch sizes, respectively. For constant batch size, we set the size to 20% of the total parameters. While for log-scale batch size, we set it to half of the remaining un-quantized parameters. We discuss five quantization set sizes with $m = 0$, 1, 2, 3, 4 and $k = -m$ (see Equation (7)). The evaluations of the three applications are presented in Sections 4.1.1–4.1.4, and the detailed result discussion is given in Section 4.1.5. The parameters of PSO algorithm in Equation (6) is shown in Table 2. The object function designed according to applications will be discussed in the following sections.

Table 2. Configuration of PSO Algorithm

N	c_1	c_2	w	$Iteration$	min_d	max_d
10	1.4	1.2	0.8	500	-2^m	2^m

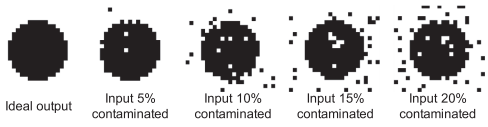
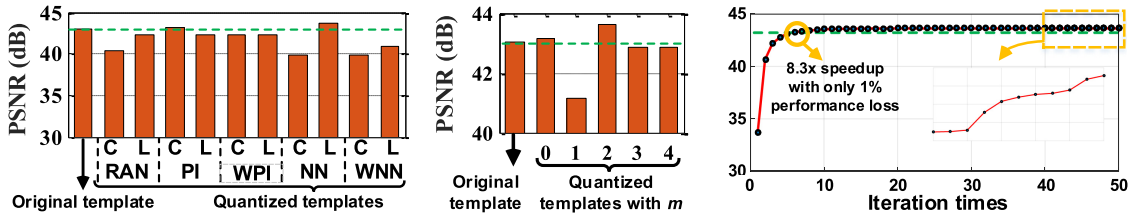


Fig. 8. Training images for **binary image noise cancellation**.



(a) m=2, k=-2 (b) NN-L, k=-m (c) Early exit

Fig. 9. Performance comparison between templates with various (a) strategies, (b) quantization sizes m, and (c) performance of early exit with NN-L strategies for for **binary image noise cancellation**.

4.1.1 Binary Image Noise Cancellation. The objective function for binary image noise cancellation in PSO re-training is shown in Equation (12), where *output* and *Ideal-Output* are output images of CeNN processing on input images with noise and desired output images, respectively, and t is the number of training pairs. The pattern structures of the 3×3 templates A and B are as follows: $A = \{0, a_0, 0; a_0, a_1, a_0; 0, a_0, 0\}$, and $B = \{a_2, a_3, a_2; a_3, a_4, a_3; a_2, a_3, a_2\}$. The training images are corrupted with salt and pepper noise as shown in Figure 8, where different levels of salt and pepper noise are added to the ideal input image. The test images are from Hlevkin test images collection [10], and gray images are transformed to binary format with contaminations of 5%, 10%, 15% and 20% salt and pepper noises. The peak signal-to-noise ratio (PSNR) is used to evaluate the quality of the processed images:

$$obj = \sum_{i=1}^{t} (output_i - IdealOutput_i)^2. \tag{12}$$

We fix the quantization size using $m = 2$ and $k = -m$, and evaluate all 10 incremental quantization frameworks. The results are depicted in Figure 9(a). From the figure, we can observe that the quantized templates achieve similar PSNR compared with the original template without quantization. The lowest PSNR is only 3dB lower than that with the original templates. Interestingly, the highest PSNR is achieved with NN-L strategy, which has an even better performance than the original template. Note that generally PI strategy achieves the best performance for CNNs [40]. However, NN-L strategy obtains the best performance for CeNN in binary image noise cancellation application. Early exit is performed with the NN-L strategy as shown in Figure 9(c). It can be noted that the performance increases very quickly in the first several iterations, and remains almost constant when the iteration times is lager than a threshold. Observe that a speedup of 8.3x is achieved with only 1% performance loss. The optimal templates (NN-L strategy) and the original templates are shown in Figure 10, and their detailed comparisons on the 20 test images are also presented. It can be observed that the PSNR of the optimal templates remains higher than that of

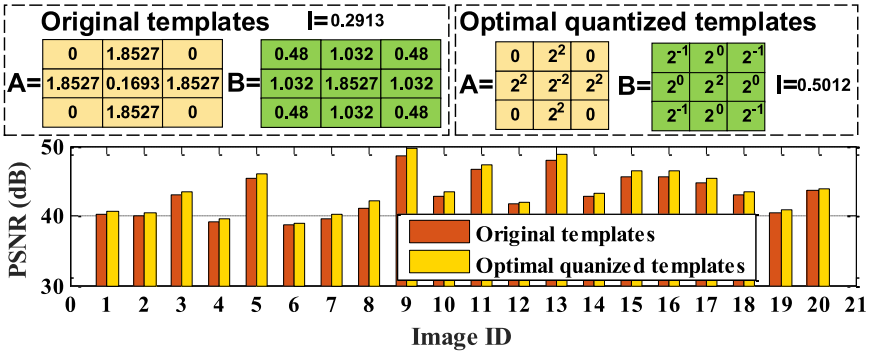


Fig. 10. Performance comparison between the optimal quantized templates and the original templates for **binary image noise cancellation**. The image ID and the image correspond as follows: (1, airfield), (2, barbara), (3, boats), (4, bridge), (5, cablecar), (6, camera), (7, cornfield), (8, fingerprint), (9, flower), (10, fruits), (11, girl), (12, goldhill), (13, lena), (14, man), (15, monarch), (16, pens), (17, pepper), (18, sailboat), (19, soccer), (20, yacht).

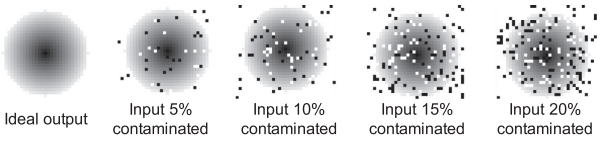
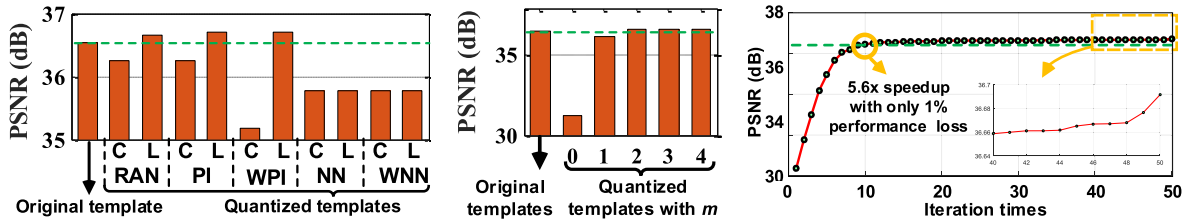


Fig. 11. Training images for **grey image noise cancellation**.



(a) m=2, k=-2 (b) PI-L, k=-m (c) Early exit

Fig. 12. Performance comparison between templates with various (a) strategies and (b) quantization sizes m, and (c) performance of early exit with PI-L strategies for **grey image noise cancellation**.

the original template across all the images. The impact of batch sizes is presented in Figure 9(b) with the optimal partition NN-L. No distinct tendency exists between PSNR and m, and note that even with $m = 0$ corresponding to the quantization set with only three values $(-1, 0, 1)$, we can still achieve a higher PSNR than that with the original templates without quantization.

4.1.2 Grey Image Noise Cancellation. The configuration for grey image cancellation is the same as that for binary image noise cancellation. The pattern structures of the 3×3 Delayed CeNN templates A, B and D are as follows: $A = \{0, 0, 0; 0, a_0, 0; 0, 0, 0\}$, $B = \{a_1, a_1, a_1; a_1, a_1, a_1; a_1, a_1, a_1;\}$, and $D = \{a_2, a_2, a_2; a_2, 0, a_2; a_2, a_2, a_2;\}$. The training images are shown in Figure 11.

The same setting of quantization with binary image noise cancellation is used, and the results are depicted in Figure 12(a). From the figure, we can note that the quantized templates still achieve similar PSNR compared with the original template without quantization. The lowest PSNR this time is only 1.5dB lower than that with the original templates. The highest PSNR is achieved with PI-L and WPI-L, both resulting in the same quantized template with an even better performance than the original template. In this application, interestingly the best strategy is PI-L, the same as that in CNNs. In Figure 12(c), we can find that the performance has the same trend with that

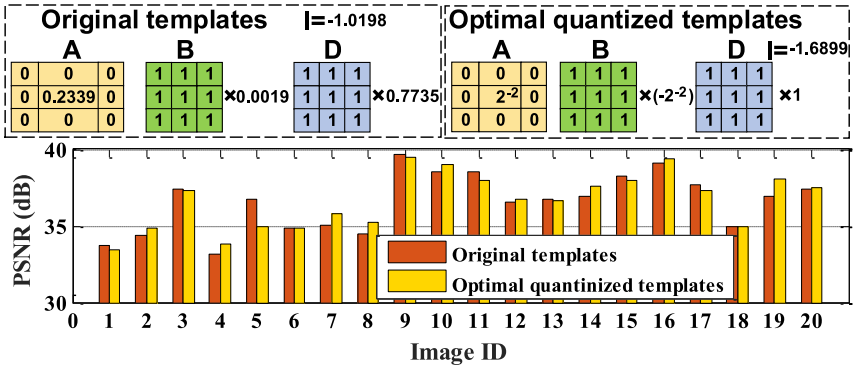


Fig. 13. Performance comparison between the optimal quantized templates and the original templates for **grey image noise cancellation**. See the caption of Figure 10 for details of image ID.

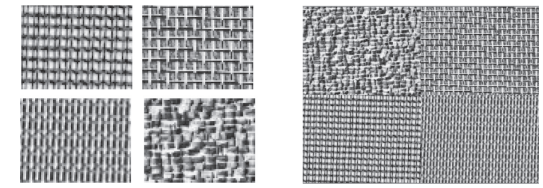


Fig. 14. Training and testing images for **texture segmentation**.

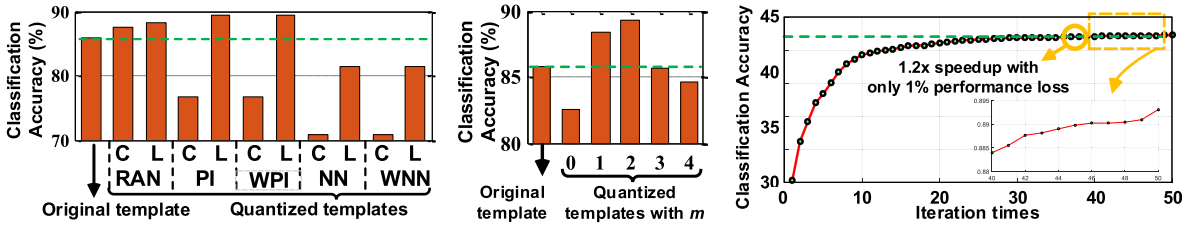
in Section 4.1.1, and early exit for the PI-L strategy can achieve a speedup of 5.6× with only 1% performance loss. The optimal templates (PI-L/WPI-L strategy) with the highest PSNR and the original templates are shown in Figure 13, and their detailed comparisons on the 20 test images are also presented. Note that the optimal templates cannot always get a higher PSNR than the original templates for the 20 images. The impact of batch sizes is presented in Figure 12(b) with the optimal partition PI-L. Note that even with $m = 0$ corresponding to the quantization set with only three values $(-1, 0, 1)$, we can still achieve a high PSNR, which is about 5.2dB lower than that with the original templates.

4.1.3 Texture Segmentation. The training and testing images are shown in Figure 14. The object function adopted from Reference [27] is shown in Equations (13) and (14), where Q_k is the area of the kth texture, G_k is the average gray-scale of the kth texture in the output numbered in ascending order of gray-level, and $g_{i,j|k}$ is the local average gray-level. A window size of 35×35 is adopted to calculate $g_{i,j|k}$. The pattern structures of the 3×3 templates A and B are as follows: $A = \{a_0, a_1, a_2; a_3, a_4, a_5; a_6, a_7, a_8\}$ and $B = \{a_9, a_{10}, a_{11}; a_{12}, a_{13}, a_{14}; a_{15}, a_{16}, a_{17}; \}$.

$$obj = \left(1 - \max_k \left(\frac{1}{Q_k} \sum_{i,j|k} e_k(i,j)\right) \times \min_k (G_k - G_{k-1})\right), \tag{13}$$

$$e_k(i,j) = \begin{cases} 0 & \text{if } (G_{k-1} + G_k)/2 < g_{i,j|k} < (G_k + G_{k+1})/2; \\ 1 & \text{else.} \end{cases} \tag{14}$$

The same setting of quantization with the above two applications is used, and the results are depicted in Figure 15(a). From the figure, we can observe that the quantized templates achieve similar accuracy compared with the original templates without quantization. The lowest accuracy is about 16% lower than that with the original templates. The highest accuracy is achieved with WPI-C and WPI-L, both resulting in the same quantized templates with a better performance compared with the original templates. As shown in Figure 15(c), early exit with WPI-C/WPI-L can obtain a



(a) m=2, k=-2 (b) WPI-L, k=-m (c) Early exit

Fig. 15. Performance comparison between templates with various (a) strategies and (b) quantization sizes m, and (c) performance of early exit with PI-L strategies for **texture segmentation**.

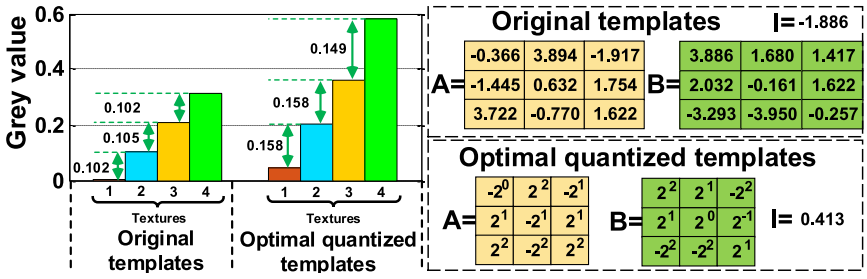


Fig. 16. Performance comparison between the optimal quantized templates and the original templates for **texture segmentation**.

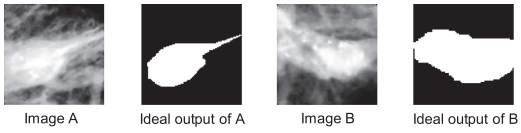
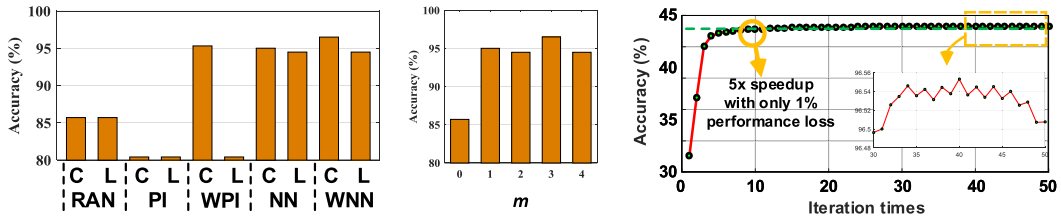


Fig. 17. Two selected images and their manually segmented images from MIAS database to train CeNN.

speedup of 1.2x with about 1% performance loss. The optimal templates (WPI-C/WPI-L strategy) for the highest accuracy and the original templates are shown in Figure 16, and their detailed comparisons are also presented. The impact of batch sizes is presented in Figure 15(b) with the optimal partition WPI-L. Note that even with $m = 0$ corresponding to the quantization set with only three values $(-1, 0, 1)$, we can still achieve a high accuracy, which is about 3.2% lower than that with the original templates.

4.1.4 Medical Image Segmentation. The objective function for medical image segmentation in PSO re-training is shown in Equation (15), where *output* and *Ideal-Output* are output images of CeNN processing on input images and desired output images, respectively, and t is the number of training pairs, and *area* is the product of the width and height of the image. We also adopts the objective function as accuracy to evaluate the quality of segmented images. The pattern structures of the 3×3 templates A and B are as follows: $A = \{a_0, a_1, a_2; a_3, a_4, a_3; a_2, a_1, a_0\}$, and $B = \{a_5, a_6, a_7; a_8, a_9, a_8; a_7, a_6, a_5\}$. The dataset is from the mammographic image analysis society (MIAS) digital mammogram database [26], and two images and its corresponding segmented results are selected as training images as shown in Figure 17, which is the of the same configuration with the work [24]. Totally 119 test images are used in the experiment. Note that as there is no ideal output in the MIAS database, the outputs of the template with double precision are regarded



(a) m=2, k=-2 (b) NN-L, k=-m (c) Early exit

Fig. 18. Performance comparison between templates with various (a) strategies, (b) quantization sizes m, and (c) performance of early exit with WNN-C strategies for **medical image segmentation**.

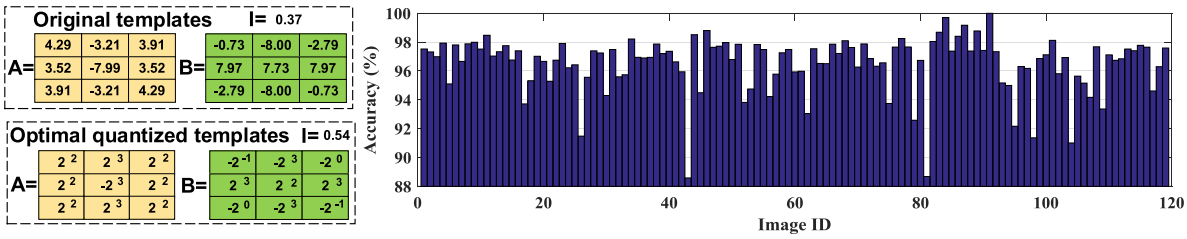


Fig. 19. Performance comparison between the optimal quantized templates and the original templates for medical image segmentation.

as the ideal outputs:

$$obj = accuracy = \sum_{i=1}^{t} abs(output_i - IdealOutput_i)/area. \tag{15}$$

As shown in Figure 18(a), we can observe that the quantized templates achieve similar accuracy compared with the original template without quantization. The lowest accuracy is about 12% lower than that with the original templates. Interestingly, the highest accuracy is achieved with WNN-C strategy, which is only 3% lower than that of the original templates. It can be interesting in the future to study this in more detail and figure out a systematic way to decide the optimal strategy. As shown in Figure 18(c), a speedup of 5× are achieved with only 1% accuracy loss for the WPI-L strategy.

The optimal templates and the original templates are shown in Figure 19, and their detailed comparisons on the 119 test images are also presented. It can be observed that the accuracy of the optimal templates has very little accuracy loss compared with the original template across almost all test images. The impact of batch sizes is presented in Figure 18(b) with the optimal partition WNN-C. The quantization set size has an interesting relationship with the performance. First, even when the quantization set is only of three values $(-1, 0, 1)$, the quantized template can still achieve high accuracy. Second, there exists an optimal m that gives the best performance and $m = 3$ for medical image segmentation. Further increasing m will not provide any performance gain.

4.1.5 Discussion. From the experiments on the four applications, it can be learned that the proposed compressed CeNN framework can generally produce quantized templates with a similar or even higher performance compared with the original templates. The optimal quantized templates for binary image noise cancellation and grey image noise cancellation can get a PSNR improvement of 0.5 and 0.1dB, respectively, while for texture segmentation, the classification accuracy is improved by 3%.

The performances of the 10 quantization frameworks for the three applications vary. It should be highlighted that unlike CNNs, the optimal strategy of CeNNs depends on applications. In terms

of parameter partition strategy, there is no clear winner that can always beat the others, and NN-L, PI-L (or WPI-L), and WPI-L (or PI-L), WNN-C can achieve the best templates for binary image noise cancellation, grey image noise cancellation, texture segmentation, and medical image segmentation, respectively. It can be interesting in the future to study this in more detail and figure out a systematic way to decide the optimal strategy. In terms of batch size, log-scale seems to perform better than constant in most cases.

The quantization set size has an interesting relationship with the performance. First, even when the quantization set is only of three values $(-1, 0, 1)$, the quantized template can still achieve high performance, which sometimes is even better than the original template (e.g., in binary image noise cancellation). Second, there exists an optimal m that gives the best performance. Further increasing m will not provide any performance gain (e.g., in texture segmentation) or may even result in performance loss (e.g., in gray image noise cancellation). This is mainly due to that a proper m will get the best balance between overfitting and underfitting for the network. The value of this optimal m depends on the detailed application and the dataset, which will also be an interesting future work.

4.2 Speed Evaluation Using FPGAs

In the previous section, we evaluated the performance of our incremental quantization framework in terms of accuracy. In this section, we will evaluate its speed when implemented in FPGAs. For a fair comparison with existing works [18, 38, 39], we adopt the same configurations of stages and try to place the maximum possible number of stages utilizing our quantized templates. Note that all the three works share the same architecture for CeNN computation. The performance of the implementation is evaluated by equivalent computing capacity, which is the product of number of stages and the computing capacity of each stage. The proposed efficient hardware implementation is implemented on an XC4LX25 FPGA. The data width of the input, state, and output (u, x, and y) is configured to be 18 bits. The widely used template size 3×3 is adopted. Note that general CeNN is adopted for the FPGA implementation, and delayed CeNN is not considered here. Time-variant templates are configured. In the implementation, multiplication is achieved with embedded multipliers (more specifically, DSP48 modules on XC4LX25 FPGAs) at first, and shifters are used when there are no more available embedded multipliers. Considering the routability of FPGAs, the utilization rate of LEs and registers are constrained to be no higher than 80%. Note that since different quantization frameworks only affects the performance and do not show significant difference in hardware resource utilization, in this part of experiments, we simply use WNN-L with $m = 5$ and $k = -5$, and other frameworks should yield almost identical speed. Note that for the same application, the iteration numbers of times are the same for fair comparison.

Three configurations of 2D convolution are discussed: one, three and nine multipliers. In Table 3, applying our quantization framework can lead to a 1.2× speedup with increased use of LEs (by 17%) and registers (by 8%). This allows an additional four stages to be placed, with a speedup of 1.2×.

Further taking sparsity-induced optimization into consideration, a speedup of 1.8× is achieved in the 2D convolution module with computations involving with template A for binary image noise cancellation. However, no sparsity exists in template B, and there is no overall speedup, as sparsity-induced optimization can only yield speedup when sparsity exists in both templates A and B. Therefore, the speedup still remain about the same. Yet after the introduction of repetition-induced optimization, the speedup can be further increased to 1.4× with slightly reduced resource usage (due to the reduction of computations needed). Note that these conclusions are application-specific. Similar conclusions reside with texture segmentation. The proposed architecture achieves a little lower clock frequency due to the high resource utilization making placement and routing relatively more difficult.

Table 3. Speed and Resource Utilization Comparisons of the State-of-the-art Work [39] and Ours with **One Multiplier/Shifter** in 2D Convolution Module, with Sparsity-induced Optimization and Repetition-Induced Optimization

Implementation	State-of-the-art (1 Multiplier)	Ours (1 Shifter)	Ours (1 Shifter+ sparsity)	Ours (1 Shifter+ repetition)
# of stages	24	28	28	24
LEs ($\times 10^3$)	14.6 (60%)	18.7 (77%)	18.7 (77%)	18.4 (76%)
Register($\times 10^3$)	8.8 (40%)	10.5 (48%)	10.5 (48%)	9.9 (46%)
Embedded Multiplier	48 (100%)	48 (100%)	48 (100%)	48 (100%)
Clock Frequency (MHz)	353	331	331	322
Cycles per pixel	11	11	11	8
Speedup	1×	**1.2×**	**1.2×**	**1.4×**

The numbers in the parentheses are the resource utilization rate.

Table 4. Speed and Resource Utilization Comparisons of the State-of-the-art Work [39] and Ours with **Three and Nine Multipliers/Shifters** in 2D Convolution Module

Implementation	State-of-the-art (3 Multiplier)	Ours (3 Shifters)	State-of-the-art (9 Multiplier)	Ours (9 Shifters)
# of stages	6	16	2	7
LEs ($\times 10^3$)	3.8 (15%)	19.6 (80%)	1.4 (5%)	18.2 (76%)
Registers ($\times 10^3$)	2.1 (10%)	6.5 (30%)	0.6 (2%)	3.6 (17%)
Embedded Multiplier	48 (100%)	48 (100%)	46 (95%)	48 (100%)
Clock Frequency (MHz)	337	320	361	343
Cycles per pixel	5	5	1	1
Speedup	1×	**2.6×**	1×	**3.5×**

The numbers in the parentheses are the resource utilization rate.

For the configuration of 2D convolution with multiple multipliers, sparsity-induced and repetition-induced optimizations doing very limited optimizations with multiple multipliers are not involved. As shown in Table 4, the state-of-the-art work in Reference [39] has a very low resource utilization (2%–15%) with LEs and registers. With the abundant resources, 10 and 5 more stages can be placed on FPGAs with shifters as a replacement of multipliers for the implementation configured with three and nine multipliers, respectively, resulting in a speedup of 2.6× and 3.5×.

As the CeNN architecture composed with stage modules are highly extensible, we make a reasonable projections to high-end FPGAs to see how the resources available in an FPGA affect the speedup. According to existing implementations on FPGAs and resource constraint of 80% LE and register utilization rate bound, the clock frequencies are assumed to be the same in the comparison. The configuration of 2D convolution with nine multipliers is adopted, which has the highest performance. We select four high-end FPGAs from Altera and Xilinx with about 500,000 to 1,000,000 LEs. As shown in Table 5, our implementations can achieve a speedup of 1.7 × −7.8×. The highest speedup of 7.8× is due to the fact that the Stratix V E FPGA has the highest rate of LEs and embedded multipliers.

5 CONCLUSIONS

In this article, we propose a compressed CeNN framework for efficient hardware implementations, which includes two parts: incremental quantization and early exit. Incremental quantization

Table 5. Speed and Resource Utilization Projections to High-end FPGAs of the State-of-the-art Work [39] and Ours with **Nine Multipliers/Shifters** in 2D Convolution Module

Implementation	VC7VX-980T	VC7VX-585T	Stratix V E	Stratix V GS
# of stages	352	179	233	291
LEs ($\times 10^3$)	780 (80%)	465 (80%)	718 (80%)	524 (80%)
Registers ($\times 10^3$)	170 (17%)	93 (16%)	133 (15%)	128 (19%)
Embedded Multiplier	3,600 (100%)	1,260 (100%)	704 (100%)	3,926 (100%)
Speedup	**2.3×**	**3.3×**	**7.8×**	**1.7×**

The numbers in the parentheses are the resource utilization rate.

adopts an iterative procedure including parameter partition, parameter quantization, and retraining to produce templates with values being powers of two. We propose a few quantization strategies based on the unique CeNN computation patterns. Thus, multiplications are transformed to shift operations, which are much more resource-efficient than general embedded multipliers. Furthermore, based on CeNN template structures, sparsity-induced and repetition-induced optimizations for quantized templates are also exploited for situations where resources are extremely limited. Early exit can fulfill the computation earlier than general computation with almost no accuracy loss. Experimental results show that incremental quantization can achieve similar or even slightly better performance compared with that using original templates without quantization, and a speedup of $1.2\times - 7.8\times$ can be achieved compared with the state-of-the-art FPGA implementations. Early exit is simple but effective, which can achieve a speedup of $1.2\times - 8.3\times$ with almost no accuracy loss. We also discover that unlike CNNs, the optimal strategy of CeNNs depends on applications. We will evaluate the performance of our method on complex and real applications as the future work.

REFERENCES

[1] Stephen J. Carey, David R. W. Barr, Bin Wang, Alexey Lopich, and Piotr Dudek. 2013. Mixed signal SIMD processor array vision chip for real-time image processing. *Analog Integr. Circ. Signal Process.* 77, 3 (2013), 385–399.

[2] Hsin-Chieh Chen, Yung-Ching Hung, Chang-Kuo Chen, Teh-Lu Liao, and Chun-Kuo Chen. 2006. Image-processing algorithms realized by discrete-time cellular neural networks and their circuit implementations. *Chaos, Solitons Fract.* 29, 5 (2006), 1100–1108.

[3] Leon O. Chua and Tamas Roska. 2002. *Cellular Neural Networks and Visual Computing: Foundations and Applications*. Cambridge University Press.

[4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv Preprint arXiv:1602.02830*.

[5] M. Duraisamy and F. Mary Magdalene Jane. 2014. Cellular neural network based medical image segmentation using artificial bee colony algorithm. In *Proceedings of the International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE'14)*. IEEE, 1–6.

[6] Osama Basil Gazi, Mohamed Belal, and Hala Abdel-Galil. 2014. Edge detection in satellite image using cellular neural network. *System* 8 (2014), 9.

[7] Hubert Harrer and Josef A. Nossek. 1992. Discrete-time cellular neural networks. *Int. Jo. Circ. Theory Appl.* 20, 5 (1992), 453–467.

[8] Hubert Harrer, Josef A. Nossek, Tams Roska, and Leon O. Chua. 1994. A current-msode DTCNN universal chip. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'94)*, Vol. 4. IEEE, 135–138.

[9] Jeremy Hills and Yongmin Zhong. 2014. Cellular neural network-based thermal modelling for real-time robotic path planning. *Int. J. Agile Syst. Manage.* 7, 3–4 (2014), 261–281.

[10] Hlevkin. 2017. Retrieved from http://www.hlevkin.com/06testimages.htm.

[11] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems*. Springer, 4107–4115.

[12] K. Karacs, G. Y. Cserey, Zarndy, P. Szolgay, C. S. Rekeczky, L. Kek, V. Szab, G. Pazienza, and T. Roska. 2007. Software library for cellular wave computing engines. *Cellular Wave Computing Library (Templates, Algorithms, and Programs)*, L. Kék, K. Karacs, and T. Roska (Eds.). Retrieved from http://cnn-technology.itk.ppke.hu/Library_v2.1b.pdf.
[13] James Kennedy. 2011. Particle swarm optimization. In *Encyclopedia of Machine Learning*. Springer, 760–766.
[14] Seungjin Lee, Minsu Kim, Kwanho Kim, Joo-Young Kim, and Hoi-Jun Yoo. 2011. 24-GOPS 4.5mm^2 digital cellular neural network for rapid visual attention in an object-recognition SoC. *IEEE Trans. Neural Netw.* 22, 1 (2011), 64–73.
[15] Huaqing Li, Xiaofeng Liao, Chuandong Li, Hongyu Huang, and Chaojie Li. 2011. Edge detection of noisy images based on cellular neural networks. *Commun. Nonlin. Sci. Numer. Simul.* 16, 9 (2011), 3746–3759.
[16] Dilan Manatunga, Hyesoon Kim, and Saibal Mukhopadhyay. 2015. SP-CNN: A scalable and programmable CNN-based accelerator. *IEEE Micro* 35, 5 (2015), 42–50.
[17] Gabriele Manganaro, Paolo Arena, and Luigi Fortuna. 2012. *Cellular Neural Networks: Chaos, Complexity and VLSI Processing*, Vol. 1. Springer Science & Business Media.
[18] J. Javier Martnez, Javier Garrigs, Javier Toledo, and J. Manuel Ferrndez. 2013. An efficient and expandable hardware implementation of multilayer cellular neural networks. *Neurocomputing* 114 (2013), 54–62.
[19] Jens Muller, Robert Wittig, Jan Muller, and Ronald Tetzlaff. 2016. An improved cellular nonlinear network architecture for binary and greyscale image processing. *IEEE Trans. Circ. Syst. II: Express Briefs* (2016).
[20] Reid Porter, Jan Frigo, Al Conti, Neal Harvey, Garrett Kenyon, and Maya Gokhale. 2007. A reconfigurable computing framework for multi-scale cellular image processing. *Microprocess. Microsyst.* 31, 8 (2007), 546–563.
[21] Sasanka Potluri, Alireza Fasih, Laxminand Kishore Vutukuru, Fadi Al Machot, and Kyandoghere Kyamakya. 2011. CNN based high performance computing for real time image processing on GPU. In *Proceedings of the Joint 3rd International Workshop on Nonlinear Dynamics and Synchronization (INDS'11) & 16th International Symposium on Theoretical Electrical Engineering (ISTET'11)*. IEEE, 1–7.
[22] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*. Springer, 525–542.
[23] Angel Rodrguez-Vzquez, Gustavo Lin-Cembrano, L. Carranza, Elisenda Roca-Moreno, Ricardo Carmona-Galn, Francisco Jimnez-Garrido, Rafael Domnguez-Castro, and S. Espejo Meana. 2004. ACE16k: The third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs. *IEEE Trans. Circ. Syst. I: Reg. Papers* 51, 5 (2004), 851–863.
[24] Rahimeh Rouhi, Mehdi Jafari, Shohreh Kasaei, and Peiman Keshavarzian. 2015. Benign and malignant breast tumors classification based on region growing and CNN segmentation. *Expert Syst. Appl.* 42, 3 (2015), 990–1002.
[25] Han Song, Pool Jeff, Tran John, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of the 4th International Conference on Learning Representations*.
[26] John Suckling, J. Parker, D. Dance, S. Astley, I. Hutt, C. Boggis, I. Ricketts, E. Stamatakis, N. Cerneaz, S. Kok et al. 1994. The mammographic image analysis society digital mammogram database. In *Exerpta Medica. International Congress Series*, Vol. 1069. 375–378.
[27] Tams Szirnyi and Mrton Csapodi. 1998. Texture classification and segmentation by cellular neural networks using genetic learning. *Comput. Vision Image Understand.* 71, 3 (1998), 255–270.
[28] Darrell Whitley. 1994. A genetic algorithm tutorial. *Stat. Comput.* 4, 2 (1994), 65–85.
[29] Henry Wong, Vaughn Betz, and Jonathan Rose. 2011. Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 5–14.
[30] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4820–4828.
[31] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. 2018. Scaling for edge inference of deep neural networks. *Nature Electron.* 1, 4 (2018), 216.
[32] Xiaowei Xu, Feng Lin, Aosen Wang, Xinwei Yao, Qing Lu, Wenyao Xu, Yiyu Shi, and Yu Hu. 2018. Accelerating dynamic time warping with memristor-based customized fabrics. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 37, 4 (2018), 729–741.
[33] Xiaowei Xu, Feng Lin, Wenyao Xu, Xinwei Yao, Yiyu Shi, Dewen Zeng, and Yu Hu. 2018. MDA: A reconfigurable memristor-based distance accelerator for time series mining on data centers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).
[34] Xiaowei Xu, Qing Lu, Tianchen Wang, Jinglan Liu, Yu Hu, and Yiyu Shi. 2017. Efficient hardware implementation of cellular neural networks with powers-of-two based incremental quantization. In *Proceedings of the Neuromorphic Computing Symposium*. ACM, 1.
[35] Xiaowei Xu, Qing Lu, Tianchen Wang, Jinglan Liu, Cheng Zhuo, Xiaobo Sharon Hu, and Yiyu Shi. 2017. Edge segmentation: Empowering mobile telemedicine with compressed cellular neural networks. In *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 880–887.

[36] Xiaowei Xu, Qing Lu, Lin Yang, Sharon Hu, Danny Chen, Yu Hu, and Yiyu Shi. 2018. Quantization of fully convolutional networks for accurate biomedical image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'18)*. IEEE/CVF, 1.

[37] Xiaowei Xu, Dewen Zeng, Wenyao Xu, Yiyu Shi, and Yu Hu. 2017. An efficient memristor-based distance accelerator for time series data mining on data centers. In *Proceedings of the 54th Annual Design Automation Conference*. ACM, 58.

[38] Nerhun Yildiz, Evren Cesur, Kamer Kayaer, Vedat Tavsanoglu, and Murathan Alpay. 2015. Architecture of a fully pipelined real-time cellular neural network emulator. *IEEE Trans. Circ. Syst. I: Reg. Papers* 62, 1 (2015), 130–138.

[39] Nerhun Yildiz, Evren Cesur, and Vedat Tavsanoglu. 2016. On the way to a third-generation real-time cellular neural network processor. In *Proceedings of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'16)*.

[40] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Toward lossless CNNs with low-precision weights. In *Proceedings of the 5th International Conference on Learning Representations*.

Received December 2017; revised April 2018; accepted August 2018