A Multi-Level-Optimization Framework for FPGA-Based Cellular Neural Network Implementation

ZHONGYANG LIU and SHAOHENG LUO, Zhejiang University
XIAOWEI XU and YIYU SHI, University of Notre Dame
CHENG ZHUO, Zhejiang University

Cellular Neural Network (CeNN) is considered as a powerful paradigm for embedded devices. Its analog and mix-signal hardware implementations are proved to be applicable to high-speed image processing, video analysis, and medical signal processing with its efficiency and popularity limited by smaller implementation size and lower precision. Recently, digital implementations of CeNNs on FPGA have attracted researchers from both academia and industry due to its high flexibility and short time-to-market. However, most existing implementations are not well optimized to fully utilize the advantages of FPGA platform with unnecessary design and computational redundancy that prevents speedup. We propose a multi-level-optimization framework for energy-efficient CeNN implementations on FPGAs. In particular, the optimization framework is featured with three level optimizations: system-, module-, and design-space-level, with focus on computational redundancy and attainable performance, respectively. Experimental results show that with various configurations our framework can achieve an energy-efficiency improvement of 3.54× and up to 3.88× speedup compared with existing implementations with similar accuracy.

CCS Concepts: • **Hardware → Reconfigurable logic and FPGAs**; *Emerging architectures*;

Additional Key Words and Phrases: Cellular neural network, FPGA, acceleration

ACM Reference format:
Zhongyang Liu, Shaoheng Luo, Xiaowei Xu, Yiyu Shi, and Cheng Zhuo. 2018. A Multi-Level-Optimization Framework for FPGA-Based Cellular Neural Network Implementation. *J. Emerg. Technol. Comput. Syst.* 14, 4, Article 47 (November 2018), 17 pages.
https://doi.org/10.1145/3273957

1 INTRODUCTION

With an increasing popularity of smart sensing and growing low-power demands for IoT applications, conventional image processing designs suffer from inefficient data processing and high-power consumption [1, 2]. Cellular Neural Network (CeNN) is considered as a viable option to improve the efficiencies in both power and performance. CeNN is a nonlinear cellular processor array inspired by the functionality of neurons through modeling the working principles of human

This article is an expanded version of our previous paper, which has been published as a conference paper in Neuromorphic Computing Symposium, Knoxville, Tennessee, July 2017 with the title "A Multi-Level-Optimization Framework for FPGA-Based Cellular Neural Network Implementation."

Authors' addresses: Z. Liu, S. Luo, and C. Zhuo, Zhejiang University: No. 38 Zheda Road, Hangzhou, China 310027; emails: {isee_lzy, shluo, czhuo}@zju.edu.cn; X. Xu and Y. Shi, University of Notre Dame: Notre Dame, IN 46556 USA; emails: {xxu8, yshi4}@nd.edu.

1550-4832/2018/11-ART47 $15.00
https://doi.org/10.1145/3273957

brain sensing. It has widely applied to noise cancellation [3], edge detection [4], path planning [5], segmentation [6, 7], and so on. Recently, CeNNs receive increasing interests from both academia and industry [8–10] for efficient hardware implementations.

CeNNs with fine granularity have the potential for analog implementations. Many studies have discussed possible analog design details [11–14] to achieve high performance with fast convergence rate and the convenience of integration with image sensors. However, due to real-time response constraints and complexity from various application scenarios, it is not a trivial task to achieve a good hardware implementation of CeNN with efficiencies in both performance and power. There are a few key challenges that need to be well addressed in the conventional CeNN analog implementations:

- **I/O Resource Consumption:** Each input is required to correspond to a unique neuron cell, consuming too many I/O resources. For example, a recent implementation in Reference [14] can only support 256×256 pixels, which is far below the demands from mainstream images specification, e.g., 1920×1080 pixels.
- **Noise Susceptibility:** Analog circuits are prone to noise, thereby limiting the output data precision to 7 bits or below [15]. Thus, such analog implementations can hardly process regular 8-bit gray images.

In view of the aforementioned issues, digital implementations of CeNNs have become a viable option [10, 16] at the cost of data approximation. Their efficiencies are limited due to the increased number of iterations for discrete approximation. For example, to process an image of 1,920×1,080 pixels, it requires 4–8 Giga operations (for 3×3 templates with around 39 operations per pixel and 50–100 iterations), which can hardly be completed to meet the typical real-time video streaming requirement of 40ms.

Recently, CeNN accelerations on digital platforms such as ASICs [10, 16], GPUs [17], and FPGAs [9, 15, 18–23] have been widely studied. Among them, FPGA is a popular alternative due to its high flexibility and low time-to-market. References [22, 23] propose low-power techniques and parallel strategies for energy-efficient neuromorphic designs. Reference [18] presented a baseline design of FPGA implementation of CeNN for several common image processing applications. Reference [19] took the advantage of reconfigurable computing for CeNNs. The proposed implementation supported multi-scale cellular image processing as well as several pattern recognition applications. Recently, a CeNN implementation for binary image processing was demonstrated in Reference [21], which focused on highly efficient processing on binary operations to achieve a 4.43× speedup. Expandable and pipelined implementations were also proposed on multiple FPGAs [20] to further reduce the computational cost. Based on that, Reference [15] implemented a high throughput CeNN system to achieve real-time video streams processing at a pixel rate of 124.4Mpixel/s. With all these works sharing the same architecture for CeNN, there still lacks a comprehensive study and design exploration framework for efficient CeNN implementation, which covers the following remaining issues:

- First, many existing works only employ a single processing element to compute one iteration, and hence do not exploit the full potential of **parallelism** [15].
- Second, many works are unaware of the repeated parameters in the template and incur unnecessary I/O overheads to obtain those parameters. Since I/O access is much slower than computation, such unawareness effectively leads to computational **redundancy**.
- Last but not least, many design-space explorations do not well study the utilization of **available resources** and **bandwidth** of FPGAs to achieve the best performance [24].

Thus, in this article, to address those remaining concerns, we propose a multi-level-optimization framework for CeNN computation for scalability and efficiency. The framework is featured with three-level optimizations:

- **System-level optimization (SLO):** A parallel and tilling processing (PTP) and a data-reuse optimization are proposed to enable the parallelism of CeNNs.
- **Module-level optimization (MLO):** For processing elements, parameter quantization is adopted to eliminate unnecessary multiplications. Moreover, a memory access scheme is proposed to make full use of memory bandwidth and reduce processing latency.
- **Design-space-level optimization (DSLO):** With different available logical resources and bandwidth, the system may achieve different performance, which, however, is not linear with resource and bandwidth, and hence cannot be simply predicted. Thus, we propose a roofline model based method to conduct more rigorous performance optimization for specified FPGAs [24].

Experiments with different configurations on multiple FPGA devices are investigated. The experimental results show that, compared with existing works, the proposed optimization framework can achieve an energy-efficiency improvement of 3.54× and a speedup of 3.88×. We then apply the proposed framework for mass segmentation to the problem of benign and malignancy diagnosis in mammogram. It shows that the CeNN based framework can achieve a better accuracy of 94.87%, compared with prior works in References [25–29], with significantly improved performance and hardware overheads.

The remainder of the article is organized as follows. Section 2 introduces background information of the article. Section 3 presents the optimized architecture. Section 4 describes system-level optimization for efficient implementation. The proposed module-level-optimization techniques are presented in Section 5. Design-space-level optimization is discussed in Section 6. Experiment and result discussion are presented in Section 7, followed by the concluding remarks in Section 8.

2 BACKGROUND

2.1 Cellular Neural Networks

CeNN model is inspired by the functionality of visual neurons, and a mass of neuron cells are connected with neighbouring 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. For the widely used 2D CeNN with space-invariant templates, the dynamics of each cell state with an M×N rectangular cell array [30] 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 forward Euler approximation as shown in Equations (3), (4), and (5).

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

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

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

Though it is possible to implement the computing operation with about 39 times operations in each iteration, $x_{i,j}$, $u_{i,j}$, and $y_{i,j}$ require extra buffers to store and transfer for subsequent operation. One solution is the Full Signal Range (FSR) model of CeNN, which is more efficient to implement by eliminating the intermediate variable $x_{i,j}$. Please refer to Reference [15] for the detailed description of the FSR model, and the modified equations are

$$y_{i,j}(n+1) = f\left(x_{i,j}(n) + \sum_{k,l=-N}^{N} A_{k,l}(n)y_{i+k,j+l}(n) + w_{ij}\right), \quad (6)$$

with the offset term

$$w_{i,j} = \sum_{k,l=-N}^{N} B_{k,l}u_{i+k,j+l} + I. \quad (7)$$

As for hardware implementation, the conversion from $x_{i,j}(n)$ to $y_{i,j}(n)$ makes it no longer necessary to store $x_{ij}(n)$ in registers any more. In this way, $y_{ij}(n)$ is limited between 1 and -1 and can be expressed in fixed bit width, which is usually 8 bits including a signal bit. Moreover, FSR model also simplifies the iteration operation. Outputs of $x_{ij}(n)$ and $w_{ij}(n)$ are used for the next iteration to calculate $y_{ij}(n+1)$ while discarding $x_{ij}(n+1)$.

For efficient hardware implementation, the discrete-time FSR CeNN model is widely adopted in FPGA implementations [9, 15, 19–21]. In particular, the computation is regular and reproducible in each iteration. To reduce the computational complexity, Equations (7) and (6) can be rewritten to Equations (8) and (9). The computation flow is divided into Process A and Process B, each of which corresponds to an iteration unit. Obviously, Process A only executes Equation (7) while Process B executes Equation (6). By exploiting the FSR model, $w_{i,j}$ only needs to be calculated for one time through all iterations. Consequently, the computation flow can be implemented as a fully pipelined architecture, and pipeline depth is equal to the total number of Euler iterations desired:

$$Process\ A: w_{i,j} = \sum_{k,l=-N}^{N} B_{k,l}u_{i+k,j+l} + I_{i,j}, \quad (8)$$

$$Process\ B: y_{i,j}(1) = \sum_{k,l=-N}^{N} A_{k,l}y_{i+k,j+l}(0) + w_{i,j}, \quad (9a)$$

$$Process\ B: y_{i,j}(2) = \sum_{k,l=-N}^{N} A_{k,l}y_{i+k,j+l}(1) + w_{i,j}, \quad (9b)$$

$$\cdots\cdots$$

$$Process\ B: y_{i,j}(N) = \sum_{k,l=-N}^{N} A_{k,l}y_{i+k,j+l}(N-1) + w_{i,j}. \quad (9c)$$

Template learning is a widely studied and applied method to find satisfactory templates $A_{k,l}$ and $B_{k,l}$ for CeNN-based applications, in which Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) are mainly used. A training algorithm based on PSO is applied to improve overall accuracy [26]. PSO searches the solution space with multiple particles in a heuristic way. Inspired by the social behavior of animals, the position update of each particle is influenced by its past local

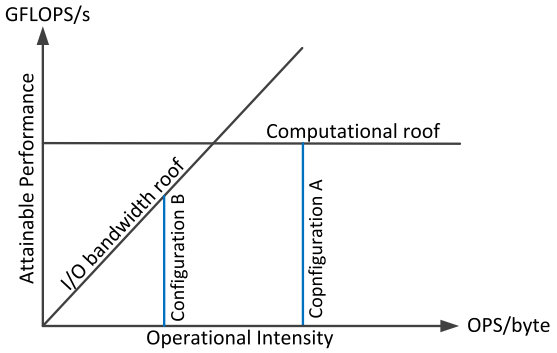


Fig. 1. Illustration of the roofline model.

best position. It performs position update and object function calculation iteratively until all the required template parameters are trained.

2.2 The Roofline Model

Figure 1 shows the roofline model based on I/O bandwidth and computational performance. The performance is bounded by the I/O bandwidth (BW) and the computational roof (CR). GFLOPS stands for the number of floating-point operations per second. Operational intensity (OI) features the number of operations per byte memory accessed, and the product of OI and BW constitutes the I/O bandwidth roof. Generally, BW and CR are considered as limited factors defined by the implementation details or obtained through benchmarks. Accordingly, the attainable performance is formulated as

$$AP = min(CR, OI \times BW). \tag{10}$$

As depicted in Figure 1, OI is relatively high in Configuration A, the applications do not need to consume the entire I/O bandwidth, and the systems are computation bound. As for Configuration B, BW becomes the limited factor, and the operation latency executed per byte are not enough to fit memory latency. CR is usually fixed for non-programmable hardware, while it becomes various for programmable FPGAs. Thus, the designs located at the right side of or just on the roofline are supported by the platform. In roofline models for FPGAs, CR, and OI are correlated. There exists a variety of configurations for a specific system, and the optimal configuration is determined by the system and the adopted hardware.

3 FRAMEWORK OVERVIEW

As shown in Figure 2, the designed framework is composed of external memory, memory interface controller, on-chip input and output buffers, a computation acceleration unit (CAU), and AXI4 bus. Due to on-chip resource limitation, data are stored in external memory and cached in on-chip buffers before processed in CAU. The workload is divided in the temporal domain with a fully pipelined structure, where different iterations are processed simultaneously in different iteration units (IUs). CAU is carried out by a chain of IUs, which is described in detail as follows:

- Iteration Unit A (IUA): The processor chain in CAU begins with IUA, which calculates both Equations (8) and (9a). As the template A_{ij} from Template RAM is constant in a time-invariant model, the result of Equation (8) remains the same in all Euler iterations. Thus, only one IUA is implemented at the very beginning. Input y of IUA can be ignored, and the first iteration feedback $y_{i,j}(0)$ is usually initialized as zero.

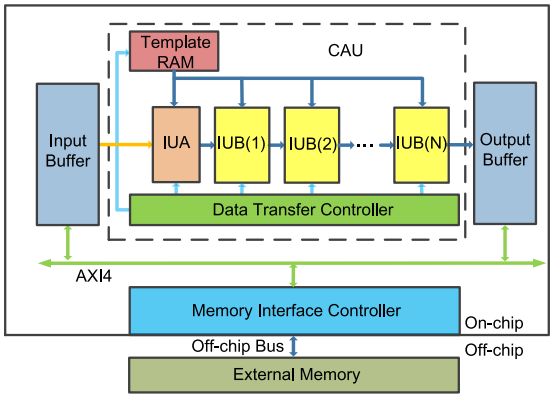


Fig. 2. Overview of the proposed optimization framework.

- Iteration Unit B (IUB): IUB indicates a number of identical modules, which form the second to the last unit in CAU. The first IUB, which is applied to calculate Equation (9b), captures $w_{i,j}$ and $y_{i,j}(1)$ from IUA. CeNN iterations are processed in a fully-pipelined architecture, i.e., the computation flow has to be divided into n+1 iterations in temporal domain before finally transferred into the output buffer.
- Data Transfer Controller: Data transfer controller manages the data transmission among the input buffer and the output buffer, template RAM and FIFOs in pipelined IUs. It also guarantees the synchronization of data transmission and computation in case of variant bandwidth features of the off-chip bus.
- Template RAM: The weights in CeNNs are space-invariant, which means that template parameters are constant between consecutive images. Therefore, Template RAM needs only a few hardware resources for storage and memory access.

The architecture is designed to load image stream from external memory, process it with CeNN and convert the result back, where CAU constitutes the most important component. The following works are based on the optimization of CAU, which will be discussed in Sections 4, 5, and 6.

4 SYSTEM-LEVEL OPTIMIZATION

In this section, the parallel and tiling processing strategies and data-reuse techniques are proposed.

4.1 Tiling Optimization

Parallel and tiling processing (PTP) is an effective approach to fully utilize computation resources and increase the maximum pixel rate. N-PTP divides input data into N parallel arrays, of which N is defined as tile size. By separating workloads in spatial domain, PTP can be implemented in two strategies, basic strategy and advanced strategy, which allocate IUs in different structures resulting in different internal memory access operations. These two PTP strategies are explained explicitly as follows:

- **Basic Strategy:** As shown in Figure 3(a), input images are divided into four parallel arrays in spatial domain for 4-PTP and are distributed into four processing elements (PEs). PE executes the computation of one pixel in one cycle. By dividing address of the input buffer, IU topology is relatively easy to implement. However, the number of DRAM ports is the same as the tile size, which brings a challenge for limited hardware resources.

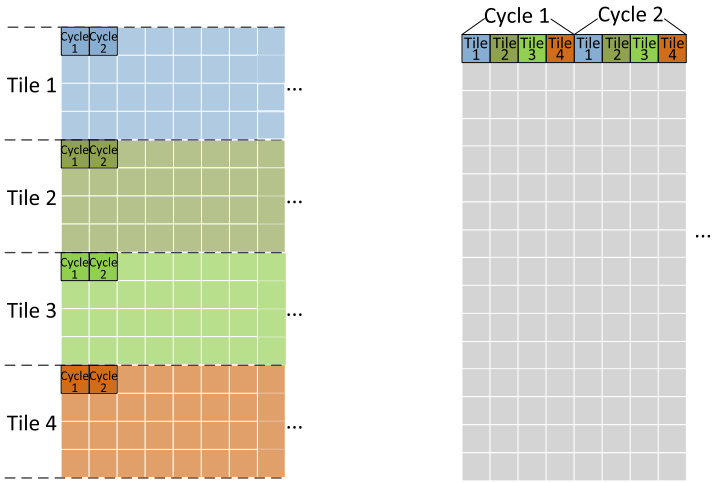


Fig. 3. Memory usage with 4-PTP for (a) basic strategy and (b) advanced strategy.

Table 1. Data Sharing Relationships of Matrix Data (Sharing with the Previous Pixel (SWP), Sharing with the Latter Pixel (SWL), and Independent Pixel (Independent))

	Column 1	Column 2	Column 3
Pixels at TOL	Independent	SWP/SWL	SWL
Pixels at EOL	SWP	SWP/SWL	Independent
Other Pixels	SWP	SWP/SWL	SWL

- **Advanced Strategy:** In Figure 3(b), input images are split into separated data chunks, of which the capacity is equal to the tile size. Data in each data chunk are arranged spatially with continuity and operated in one cycle. A data chunk occupies data by the least amount of DRAM ports when meeting the condition of maximum write/read bandwidth. Therefore, the number of DRAM ports decreases, which reduces hardware resources overheads.

4.2 Data-reuse Optimization

As shown in Equation (3), the computation of CeNN is accompanied by two 3×3 matrices Y and U, which repeats in a regular manner in adjacent pixels. Data-reuse optimization is to minimize the memory access operation by utilizing repeated data. The distribution and location of repeated data determine how the algorithm operates. As shown in Table 1, data sharing relationships depend on the location of pixels. Except for pixels at the top of line (TOL) and the end of line (EOL), the other pixels have the same sharing relationships. Nine elements of a matrix are divided into three groups by column, thus all pixels are classified into three categories.

When calculations corresponding to Equations (7) and (9) are performed in IUs, data-reuse based on data sharing relationships eliminates excessive read and write operations of IUs and the input buffer, which is a promising way to simplify memory access. As an example, the hardware implementation of a 3-PTP IU block design with data-reuse technique is shown in Figure 4, with image data tiled into three parallel arrays. Note that adjacent pixels share two columns of data for

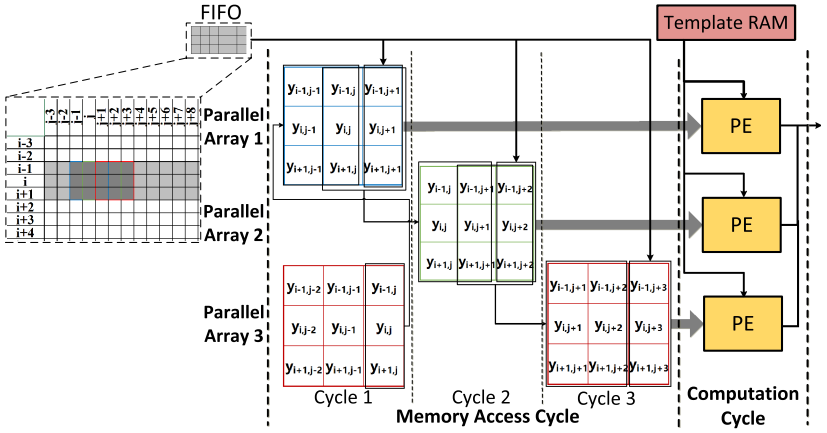


Fig. 4. Data reuse optimization in advanced 3-PTP: Two columns of data are shared for a 3×3 template, resulting in 2/3 memory access reduction.

a 3×3 template, we only need to access one column of data from FIFO, which results in 2/3 memory access reduction.

5 MODULE-LEVEL OPTIMIZATION

As for the module level, a large number of multiplications are required using the limited embedded multipliers on FPGAs, and the data access latency limits the maximum computation rate. In this section, the further design exploration is presented for IUs to overcome these shortcomings. We propose parameter quantization to reduce multiplier consumption and memory access optimization to eliminate redundant data transfer time.

5.1 Parameter Quantization

Equation (9) depicts nine multiplications for CeNN calculation, which provides more opportunities to improve the computing process. As existing works have proved, 95%–100% embedded multipliers are used while only 5% LUTs are used for a FPGA-based CeNN implementation, which becomes a bottleneck [20]. A possible solution is to decrease or eliminate multipliers required. Investigating the fact that embedded multipliers only occupy a small proportion of the core area, shifters are alternative methods for the integer operations. We propose parameter quantization to optimize computation, which performs power-of-two conversion on all template values. The quantization is depicted in Equation (11), where k and m indicate the range, and $uq(i)$ presents the optimized parameters. This technique, although has little impact on memory usage or computation complexity, reduces multiplier consumption by transferring multiplications into logic shifts:

$$uq(i) = \begin{cases} 2^p & \text{if } 3 \times 2^{p-2} \le |uq(i)| < 3 \times 2^{p-1}; \\ & \quad k \le p \le m; \\ 2^m & \text{if } |uq(i)| \ge 2^m; \\ 0 & \text{if } |uq(i)| < 2^{-k-1}. \end{cases} \tag{11}$$

It is noted that template quantization may bring two critical scenarios. First, with power-of-two parameter quantization, most of the templates occupy 5–6 repeated values, which enables repetition-induced optimization. Therefore, multiplications and additions among repeated values can be optimized, e.g., $a_1 \times y_1 + a_1 \times y_2 + a_1 \times y_3 = (a_1 + a_2 + a_3) \times y_3$, three multiplications is optimized to one. Second, parameter quantization generates a sparser matrix with some parameters

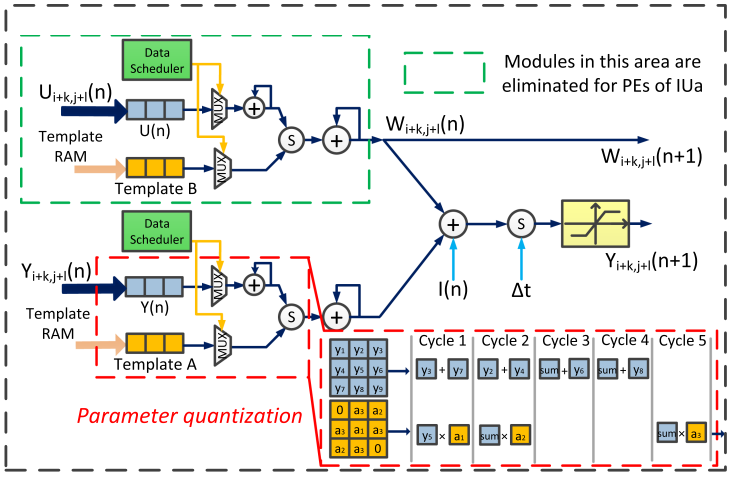


Fig. 5. Quantization optimization for CeNN in PEs: Processing time is reduced from nine cycles to five cycles, with repetition and sparsity optimization in Data Scheduler.

rounding to zero. Parameters whose absolute values are smaller than 2^{-k-1} are rounded to zero after the conversion. The quantization optimization is detailed in Figure 5. The sparsity-induced and repetition-induced optimization are found to be efficient in most CeNN applications, based on our analysis of 87 tasks with 79 applications, which shows that 78.2% templates are sparse matrix, while 94.4% templates contain repetitive parameters [31].

5.2 Memory Access Optimization

Since memory access time is mostly one order of magnitude lower than computation time, it has little impact for basic PTP strategy. However, with the increased amount of input data, the memory access time for parallel arrays increases as well when using advanced PTP strategy. Thus, the proportion of computation time slumps, and it is worthy to reorganize and manipulate the on-chip memory access and computation scheme to reduce processing latency.

Image data are buffered for each phase, while loading and offloading time for registers are independent. we divide one phase into two parts, *ComputationA* and *ComputationB*. When PE is processing for *ComputationA*, registers offload data for the current phase. While no offloading operation is required in *ComputationB*, registers are prepared to load data for the next phase in memory access cycle. Figure 6 shows the timing of several computation and memory access phases for parallel arrays. When investigating the memory access optimization in advanced PTP, we overlap parallel phases, and arrange memory access cycles in a pipelined strategy. Note that this optimization work for any data transfer mechanism in this framework, which is determined by computation cycle and memory access cycle. The phase cycle for general applications is expressed in Equation (12):

$$Phase\ cycle = max\left(memory\ access\ cycle \times \left(parallel\ size - \frac{computation\ cycle}{memory\ access\ cycle}\right), computation\ cycle\right). \tag{12}$$

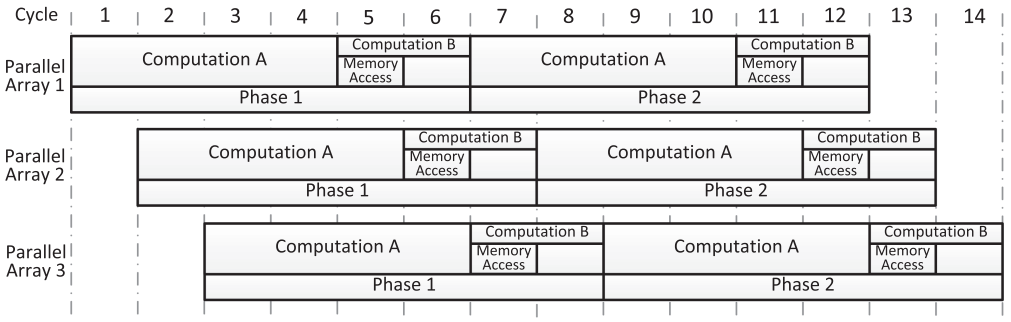


Fig. 6. Pipelined strategy of memory access and computation.

6 DESIGN-SPACE-LEVEL OPTIMIZATION

Roofline model based on I/O bandwidth and computational performance is used to evaluate AP of the target platform. As exposed in Section 5.1, floating-point operations are converted into fixed-point integer operations. The concept of byte-operations are adopted to analyze different kinds of integers exclusively, which defines a single operation of one byte integer. By classifying the operation complexity of different bit width, gigabyte operations per second (GOPS/s) is used as the metric of CR. CR_{FPGA} is defined as follows:

$$CR_{FPGA} = ((n_{IU} - 1) \times CR_{PEB} + CR_{PEA}) \times parallel\ size. \tag{13}$$

There are one IUA and n_{IU}-1 IUBs implemented in the proposed CeNN architecture in total. We present computational roof of PE in IUA (CR_{PEA}) and IUB (CR_{PEB}) to indicate different structures in two kinds of IUs, which are used as basic elements to conclude CR_{FPGA}. Parallel size reflects the number of parallel PEs in one IUB.

Moreover, OI also varies with the reconfiguration of PEs. Since the advanced PTP strategy reduces memory resource consumption and I/O ports usage, a significant improvement in OI is achieved. It is practicable to get OI:

$$OI = \begin{cases} operations_{PE} \times n_{IU} & \text{Advanced PTP} \\ \frac{operations_{PE} \times n_{IU} \times \#row}{\#row + 2 \times (parallel\ size - 1)} & \text{Basic PTP} \end{cases}. \tag{14}$$

Here $operations_{PE}$ indicates byte operations in one PE per computation cycle. AP is determined by the minimum of I/O bandwidth (BW) roof and CR, which is depicted as follows:

$$AP = min(CR_{FPGA}, OI \times BW). \tag{15}$$

In FPGA-based hardware applications, different number of AXI-IP interfaces are set to alter I/O bandwidth of AXI bus to data transfer controller. As shown in Figure 7, adding AXI-IP interfaces from 2 to 16 drives a fivefold increase in LUT consumption, and the maximum I/O bandwidth rises almost linearly from 650MB/s to 3.6GB/s. Consequently, a framework with as few AXI-IP interfaces as possible to meet the requirement of I/O bandwidth roof is the optimal one.

In Figure 8(a), applying all possible system level and module-level-optimization methods, two roofline models with different I/O bandwidth are presented. Theoretical roofline models perform the impacts that OI has on the attainable roof, with the prerequisite that BW and CR are fixed for any applications. However, roofline models for FPGA-based frameworks are not so explicit. For example, although two framework are implemented on the same platform in Figure 8(b), different I/O bandwidths result in various hardware overheads. BW1 achieves a higher I/O bandwidth roof and a lower CR compared with the counterpart of BW2, for more implemented AXI-IP interfaces

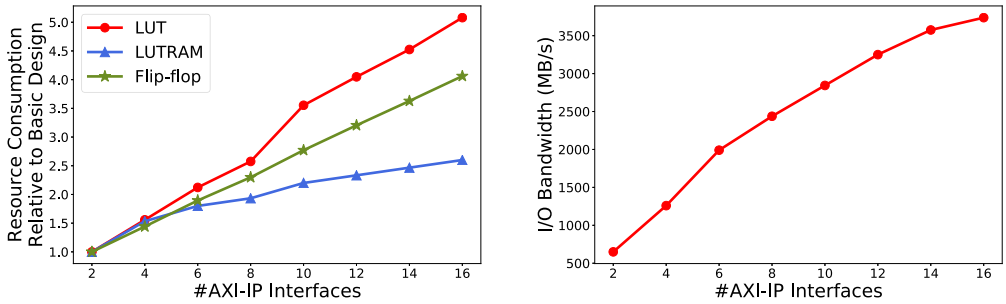


Fig. 7. (a) Resource consumption and (b) I/O bandwidth with different number of AXI-IP interfaces.

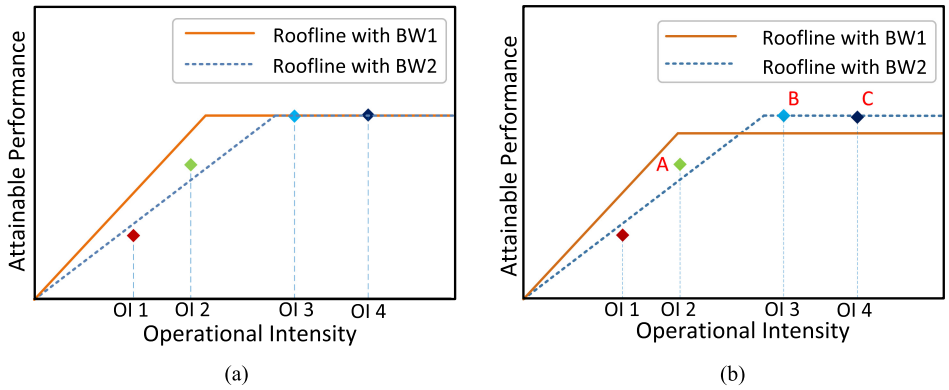


Fig. 8. Comparison of (a) theoretical and (b) FPGA-based roofline models.

with BW1 reduces the available computation resources. Therefore, B and C cannot be satisfied by the CR with BW1 at the target platform.

We explore the design space for the platform and enumerate a set of implementations with the highest attainable performance. In most cases, we could find more than one design within the set, e.g., both B and C satisfy the highest AP with BW2 in Figure 8(b). The selected design ought to be C, which derives from the fact that OI 4 is greater than OI 3 with the same AP. Therefore, fewer AXI-IP interfaces and hardware resources are required for the implementation.

7 EXPERIMENT

7.1 Multi-level Optimization

We implemented the proposed CeNN framework on Xilinx Zybo board with Zynq XC7Z010 and Zedboard with Zynq XC7Z020, with critical paths optimized based on timing reports. Implementation reports are generated to summarize the resource utilization and estimated latency accurately and efficiently. We then evaluate the proposed techniques of three-level optimizations for CeNNs with a specified number of iterations to ensure the performance comparison is conducted under the same resource constraint.

7.1.1 System-Level Optimization. Table 2 summarizes the performance comparison of the basic CeNN approach (Basic) and the implementation of the proposed framework using different optimization schemes. To partition the contribution of various level optimizations in a quantitative

Table 2. Resource Utilization and Performance Comparison Under the Constraints of 3-PTP and Eight Pipelines

	Basic [15]	SLO	MLO
LUTs	15,336	10,232	13,632
FFs	10,824	6,944	10,656
Computational Performance(GOPS/s)	5.75	5.34	9.06
Energy consumption (μJ)	2,823	733	1,366
Computational density (GOPS/LUTs)	3.75×E-4	5.22×E-4	6.65×E-4
Improvement	1×	1.39×	1.78×

way, we employ the advanced 3-PTP strategy using eight pipelines and then conduct system-level optimization (SLO) and module-level optimization (MLO). We will discuss design-space-level optimization (DSLO) with comparisons between combined optimization techniques in Section 7.1.4, for DSLO contributes a further development on the basis of SLO and MLO. The basic approach is extracted from Reference [15], which is the state-of-art CeNN hardware design. Due to its specific architecture, it is reconfigured to make the same feature size as our proposed framework for a fair comparison. We evaluate the resource consumption, energy consumption, computational performance and density.

The on-chip energy per classification to solve the DDSM segmentation problem [32] (439×454 resolution on average at 100MHz clock rate) is reported by Xilinx Vivado 2016.2. SLO achieves a significant reduction mainly because of the reuse of FIFOs. Since different architectures investigate different optimization strategies under different FPGA platforms, it is challenging to make a straightforward comparison among them. For a fair comparison, we introduce the concept of computational density as a measure of energy efficiency, which is hence independent of hardware platforms. As in Reference [33], computational density is defined as the average GOPS per area unit (GOPS/LUTs).

As shown in the first two rows of Table 2, SLO achieves a higher performance, which is 1.39× more energy efficient compared with the basic approach, while its resource utilization of LUTs decreases to 66.72%.

7.1.2 Module-Level Optimization. MLO include two key techniques, parameter quantization (PQ) and memory access optimization (MA), as discussed in Section 5. To better illustrate the impact of MLO on computational performance, we compare the efficiency by enabling PQ and MA separately as well as their combination (PQ+MA). Figure 9 shows the results. It is worthwhile to be noted that unlike basic approach, whose performance continues climbing to 12GOPS/s with nine multipliers, PQ+MA easily converges to 18GOPS/s with merely three multipliers. The convergence is due to the fact that the part exceeding 3 multipliers would be redundant for the computation in one PE. Compared with the basic approach, MA brings a significant performance improvement of 1.5×. By utilizing PQ, the minimum number of multipliers to reach peak performance is reduced from nine to three. As shown in the third row of Table 2, MLO achieves a speedup of 1.58× and an energy-efficiency improvement of 1.78× with its resource utilization decreasing to 89% compared with the basic approach.

7.1.3 Design-Space-Level Optimization. As discussed in Section 6, with specified hardware resources and I/O bandwidth, we can propose an exclusive roofline model for a specific algorithm. For different FPGA platforms, we can then obtain different roofline models accordingly of attainable performance and operational intensity. Two roofline models for two different FPGA devices

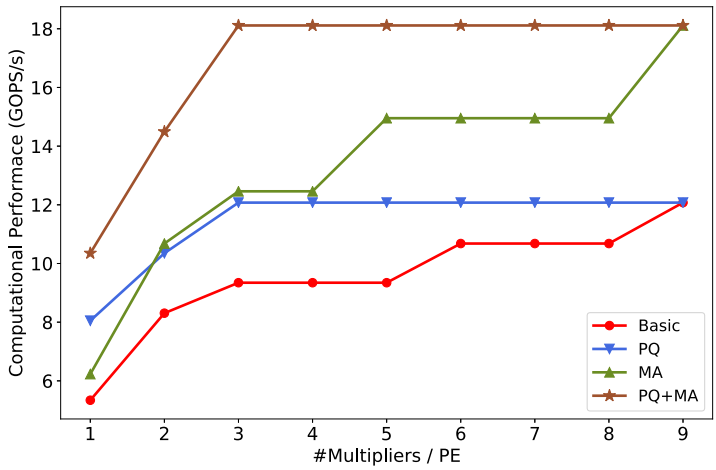


Fig. 9. Computational performance with different number of multipliers per PE.

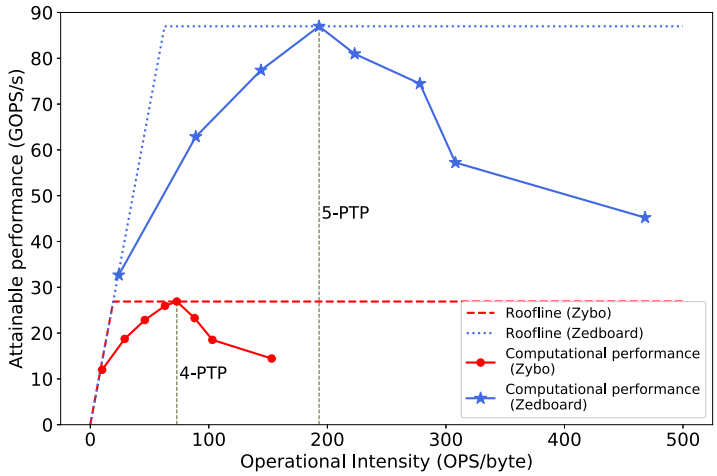
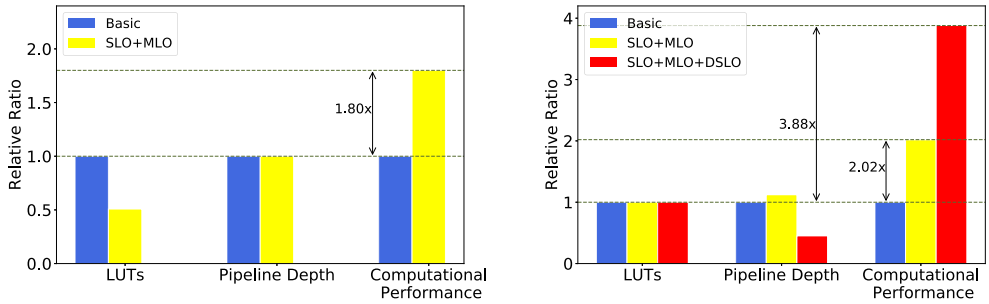


Fig. 10. Roofline models of the proposed design-space optimization on two different platforms (Zybo: 17,600 LUTs; Zedboard: 53,200 LUTs).

are presented in Figure 10. As mentioned in Section 6, given specific basic elements IUA and IUB, tile size n_{IU} and parallel size, the computational roof can be calculated by Equation (13). Then, we enumerate all the possible tile sizes and parallel sizes and generate a set of working modes with different operational intensity that the FPGA may achieve. The upper bound of attainable performance is considered as the highest computational roof among these working modes. The line of bandwidth roof is also defined by the platform specification. Any working modes at the left side or the upper side of two roofs require overmuch bandwidth or resources, respectively, which exceeds the target FPGA platform capacity. As OI increases, the attainable performance increases until it comes to a local maximum where the peak performance is obtained. After that the I/O resource consumption continues increasing and then degrades the performance. Thus, with DSLO, the optimal solution can be selected for the given platform with desired performance and operational



(a) Specified numbers of PTP and pipeline depth (b) Constrained resource utilization

Fig. 11. Pipeline depth and performance comparisons among Basic, SLO+MLO, and SLO+MLO+DSLO.

intensity. As shown in Figure 10, 4-PTP tends to be an optimal model for Zybo, while 5-PTP for Zedboard.

7.1.4 Overall Optimization. In previous subsections, we discussed the performance of the single-level optimization technique under the constraint of 3-PTP and eight pipelines. In this subsection, we investigate the efficiency of the proposed techniques, when combining different levels of optimization, under various constraint scenarios. Two key constraints are identified: (1) specified numbers of PTP and pipeline depth; (2) resource utilization. Again, similar as in Table 2, we compare the combined optimization techniques SLO+MLO and SLO+MLO+DSLO, with the basic approach (Basic). It is noted that for constraint (1) due to different resource overheads, DSLO cannot be employed under this constraint. Thus, only SLO+MLO and Basic are compared. Figure 11(a) illustrates the comparisons of LUTs, pipeline depth and performance among results from the combinations and basic approach with constraint (1). SLO+MLO achieves a 1.8× speedup and a 3.54× energy-efficiency enhancement, which can be roughly estimated by computational density compared with the basic model. Meanwhile, resource utilization of LUT decreases by 49.19% in total, which is much larger than utilizing MLO or SLO alone.

To show the contribution of DSLO quantitatively, more results are depicted in Figure 11(b), where constraint (2) is applied and pipeline depth is variable in the same FPGA platforms. The optimal model based on SLO+MLO+DSLO has the highest computational performance and the least pipeline depth. The highest computational performance achieves a 3.88× speedup.

7.2 Application for Mammographic Mass Segmentation

We also evaluate the proposed optimization framework on a real application, breast cancer segmentation, which is popular and critical for breast cancer classification and diagnosis. As shown in Figure 12, the segmentation method in the proposed system is performed on images from a widely used database, *digital database for screening mammography* (DDSM) [32].

The proposed system for benign and malignant breast tumor diagnosis consists of preprocessing, segmentation, feature extraction and tumor classification, of which we focus on the CeNN segmentation stage. For CeNN template training, particle swarm optimization (PSO) is used. We selected 161 cases containing malignant and benign masses. The intensity, textural, and shape features are then extracted from segmented tumors to reveal the characteristics of the segmented masses. The selected features are given to a Multi-layer Perceptron (MLP) as the input vectors [25].

For fair comparison, prior works using different segmentation approaches in diagnosing benign or malignant tumors on DDSM database are listed in Table 3. For example, Reference [25] uses region growing algorithm whose threshold is obtained by a trained neural network. The CeNN

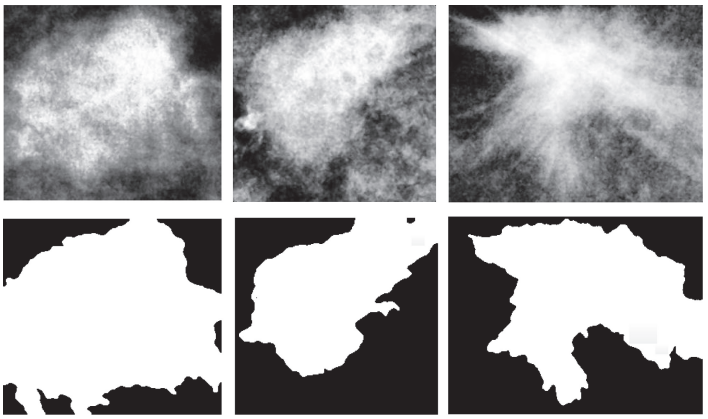


Fig. 12. Comparison between original images from DDSM database (the first row) and segmented images (the second row).

Table 3. Comparison with Our Proposed Method and Existing Methods in Benign or Malignant Tumors Diagnosis

Method	Segmentation Approach	Accuracy (%)
Proposed method	CeNN& PSO	94.87
Rahimeh et al., 2014 [25]	Region Growing	94.67
Wang and Yang et al., 2014 [26]	CeNN& DGA	92.74
Liu and Tang, 2013 [27]	Level-Set-Based	93.00
Zhang et al., 2012 [28]	Multiple Weak Segmentors	72.00
Verma et al., 2010 [29]	ANN	88.75

whose template is trained with distributed genetic algorithm (DGA) is proposed in Reference [26]. A level-set-based method that adopts the FCM clustering with spatial information constraints is employed in Reference [27]. It indicates that the CeNN segmentation method provides comparable results. The performance of classification accuracy is even better than that of prior works. Furthermore, our proposed framework can obtain a resource reduction of 49.2% and a 1.8× speedup.

8 CONCLUSIONS

We propose an efficient multi-level-optimization framework for the FPGA CeNN implementations. In system-level, parallel and tilling processing (PTP) and data-reuse optimization are applied. The computation flow is improved by separating the workload in spatial domain. In module-level optimization, we propose parameter quantization to reduce multiplier overheads and memory access optimization for the data transfer mechanism. In design-space optimization, we apply different aforementioned optimized designs and obtain roofline models accordingly of attainable performance and operational intensity. Then the optimal CeNN solution can be selected with desired performance and operational intensity. Finally, evaluations of the proposed framework are presented on different FPGA platforms, and a speed up of 3.88× is achieved compared with existing implementations. Applications for mammographic mass segmentation based on the proposed CeNN framework achieves comparable accuracy compared with existing works, while achieving a 1.8× speedup with 49.2% less resources consumption.

REFERENCES

[1] Cheng Zhuo, Kassan Unda, Yiyu Shi, and Wei Kai Shih. 2018. From layout to system: Early stage power delivery and architecture co-exploration. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* PP, 99 (2018), 1–1.

[2] Cheng Zhuo, Gustavo Wilke, Ritochit Chakraborty, Alaeddin A. Aydiner, Sourav Chakravarty, and Wei-Kai Shih. 2015. Silicon-validated power delivery modeling and analysis on a 32-nm DDR I/O interface. *IEEE Trans. Very Large Scale Integr. Syst.* 23, 9 (2015), 1760–1771.

[3] 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.

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

[5] Jeremy Hills and Yongmin Zhong. 2014. Cellular neural network-based thermal modelling for real-time robotic path planning. *Int. J. Agile Syst. Manage. 20* 7, 3–4 (2014), 261–281.

[6] 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.

[7] Zhongyang Liu, Cheng Zhuo, and Xiaowei Xu. 2018. *Efficient Segmentation Method Using Quantised and Non-linear CeNN for Breast Tumour Classification*. Electronics Letters.

[8] Fadi Al Machot, Mouhannad Ali, Ahmad Haj Mosa, Christopher Schwarzlmüller, Markus Gutmann, and Kyandoghere Kyamakya. 2016. Real-time raindrop detection based on cellular neural networks for ADAS. *J. Real-Time Image Process.* (2016), 1–13.

[9] 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 International Workshop on Cellular Neural Networks and their Applications (CNNA'16)*.

[10] Dilan Manatunga, Hyesoon Kim, and Saibal Mukhopadhyay. 2015. SP-CNN: A scalable and programmable CNN-based accelerator. *IEEE Micro* 35, 5 (2015), 42–50.

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

[12] 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.

[13] Gabriele Manganaro, Paolo Arena, and Luigi Fortuna. 2012. *Cellular Neural Networks: Chaos, Complexity and VLSI Processing*. Vol. 1. Springer Science & Business Media.

[14] 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.

[15] 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.

[16] Seungjin Lee, Minsu Kim, Kwanho Kim, Joo-Young Kim, and Hoi-Jun Yoo. 2011. 24-GOPS 4.5-mm^2 digital cellular neural network for rapid visual attention in an object-recognition SoC. *IEEE Trans. Neural Netw.* 22, 1 (2011), 64–73.

[17] 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 Nonlinear Dynamics and Synchronization (INDS) & 16th International Symposium on Theoretical Electrical Engineering (ISTET), 2011 Joint 3rd International Workshop*. IEEE, 1–7.

[18] 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.

[19] 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.

[20] J. Javier Martnez, Javier Garrigs, Javier Toledo, and J. Manuel Fernandez. 2013. An efficient and expandable hardware implementation of multilayer cellular neural networks. *Neurocomputing* 114 (2013), 54–62.

[21] 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* 65, 8 (2016), 1084–1088.

[22] Qian Wang, Youjie Li, Botang Shao, Siddhartha Dey, and Peng Li. 2017. Energy-efficient parallel neuromorphic architectures with approximate arithmetic on FPGA. *Neurocomputing* 221 (2017).

[23] Qian Wang, Yingyezhe Jin, and Peng Li. 2015. General-purpose LSM learning processor architecture and theoretically guided design space exploration. In *Proceedings of the Biomedical Circuits and Systems Conference (BioCAS'15)*. 1–4.

[24] Bruno da Silva, An Braeken, Eril H. D'Hollander, and Abdellah Touhafi. 2013. Performance modeling for FPGAs: Extending the roofline model with high-level synthesis tools. *Int. J. Reconfig. Comput.* 2013, Article 7 (2013).

[25] 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.
[26] Wei Wang, Li-Jun Yang, Yu-Ting Xie, and You-wei An. 2014. Edge detection of infrared image with CNN_DGA algorithm. *Optik-Int. J. Light Electron Optics* 125, 1 (2014), 464–467.
[27] Xiaoming Liu and Jinshan Tang. 2014. Mass classification in mammograms using selected geometry and texture features, and a new SVM-based feature selection method. *IEEE Syst. J.* 8, 3 (2014), 910–920.
[28] Yu Zhang, Noriko Tomuro, Jacob Furst, and Daniela Stan Raicu. 2012. Building an ensemble system for diagnosing masses in mammograms. *Int. J. Comput. Assisted Radiol. Surg.* 7, 2 (2012), 323–329.
[29] Brijesh Verma, Peter McLeod, and Alan Klevansky. 2010. Classification of benign and malignant patterns in digital mammograms for the diagnosis of breast cancer. *Expert Syst. Appl.* 37, 4 (2010), 3344–3351.
[30] Leon O. Chua and Tamas Roska. 2002. *Cellular Neural Networks and Visual Computing: Foundations and Applications.* Cambridge University Press.
[31] K. Karacs, G. Y. Cserey, Zarndy, P. Szolgay, C. S. Rekeczky, L. Kek, V. Szab, G. Pazienza, and T. Roska. 2010. Software library for cellular wave computing engines. Cellular Sensory and Wave Computing Laboratory of the Computer and Automation Research Institute. Hungarian Academy of Sciences (MTA SZTAKI), and the Jedlik Laboratories of the Pazmany University.
[32] M. Heath, K. Bowyer, D. Kopans, R. Moore, and P. Kegelmeyer. 2001. *The Digital Database for Screening Mammography.* Springer, Netherlands, 457–460.
[33] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* ACM, 161–170.

Received December 2017; revised May 2018; accepted August 2018