Edge Segmentation: Empowering Mobile Telemedicine with Compressed Cellular Neural Networks

Xiaowei Xu[1,2], Qing Lu[2], Tianchen Wang[2], Jinglan Liu[2]
Cheng Zhuo[3], Xiaobo Sharon Hu[2], and Yiyu Shi[2]
1 Huazhong University of Science and Technology, Wuhan, China
2 University of Notre Dame, South Bend, IN, USA
3 Zhejiang University, Hangzhou, China
yshi4@nd.edu

ABSTRACT

With the need for increased care and welfare of the rapidly aging population, mobile telemedicine is becoming popular for providing remote health care to increase the quality of life. Recently, image analysis is being actively applied for medical diagnosis and treatment, in which image segmentation is of the fundamental importance for other image processing such as visualization and detection. However, given the tasks challenges in transmitting large volume of high-resolution images and the real-time constraints that are commonly present for mobile telemedicine, image segmentation is best done at the "edge", i.e., locally so that only segmentation results are communicated. A powerful approach to medical image segmentation is cellular neural network (CeNN), which can achieve very high accuracy through proper training. However, CeNNs typically involve extensive computations in a recursive manner. As an example, to simply process an image of 1920x1080 pixels requires 4-8 Giga floating point multiplications (for 3x3 templates and 50-100 iterations), which needs to be done in a timely manner for real-time medical image segmentation. Such a demand is too high for most low power mobile computing platforms in IoTs. This paper presents a compressed CeNN framework for computation reduction in CeNNs, which is the first in the literature. It involves various techniques such as early exit and parameter quantization, which significantly reduces computation demands while maintaining an acceptable performance.

1. INTRODUCTION

The rapid aging of the world's population has come with a significant increase in the prevalence of chronic diseases and their effects, resulting with increasing need for care and welfare [14]. To tackle this burden, mobile telemedicine providing remote medical care has been popular, in which medical care services, diagnosis, consultation, treatment, and the transfer of medical data are delivered using interactive audiovisual and data communications [2].

Recently, with the improvement of the performance of medical imaging equipment, analysis on the produced high-resolution medical images is being actively applied for medical care, which can enhance effect of diagnosis and treatment by assisting the specialists. Especially, image segmentation to extract the tissue information from these images plays a key role for successful analysis, which is of fundamental importance for other image processing such as visualization and detection. Usually segmentation on these large volume of high-resolution medical images can be performed on cloud. However, due to the transmission difficulty of the large data volume and the real-time requirement, the segmentation is best done at the "edge", i.e., locally so that only segmentation results are communicated for mobile telemedicine.

A very powerful tool for medical image segmentation is cellular neural network (CeNN), which can achieve very high accuracy through proper training. It should be noted that CeNNs are popular in image processing areas such as classification [6], segmentation [7], while convolutional Neural Networks (CNNs) are most powerful in classification related tasks. However, due to the complex nature of segmentation and other image process 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 [9][22][16][1]. The advantages of analog implementations include high performance with an extremely fast convergence rate 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 requirement of mainstream images, e,g., 1920×1080 pixels. Second, analog circuits are prone to noise, which limit the output data precision to 7 bits or below [28]. 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 1920x1080 pixels requires 4-8 Giga operations (for 3×3 templates and 50-100 iterations), which needs to be done in a timely manner for real-time medical image segmentations.

To tackle the computation challenge, CeNN accelerations on digital platforms such as ASICs [12][15], GPUs [20] and FPGAs [3][19] [17][28][29] [18] have been explored, with FPGA among the most popular choices due to its high flexibility and low time-to-market. The work [3] presented a baseline design with several applications, while the study [19] took advantage of reconfigurable computing for CeNNs. Recently, the CeNN implementation for binary images was demonstrated [18]. Expandable and pipelined implementations were achieved on multiple FPGAs [17]. Taking advantage of the structure in [17], the work [28] implemented a high throughput real-time video streaming system, which is further improved to be a complete system for video processing [29]. All the three works share the same architecture for CeNN computation. Due to the large number of multiplications needed in CeNNs, the limited number of embedded multipliers in an FPGA become the bottle-

978-1-5386-3093-8/17/$31.00 ©2017 IEEE

neck for further improvement. For example, in work [17] 95%-100% of the embedded multipliers are used. On the other hand, 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 [26]. 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. 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 would not lead to significant improvement and we aim 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 values 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 [30], but as detailed in Section 2.3, the difference in computation structures between CeNNs and CNNs warrants a separate investigation for CeNNs. And indeed, our findings show 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 paper we present a compressed CeNN framework for computation reduction in CeNNs. Particularly, we systematically put forward powers-of-two based incremental quantization of CeNNs for efficient hardware implementation and early exit optimization. The 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. Out of the five only pruning-inspired strategy and random strategy have been adopted in incremental quantization of CNNs [30] due to the differences in their computation patterns. The early exit optimization can fulfill the computation earlier than general computation with almost no accuracy loss.

We have conducted extensive experiments with three widely used applications to evaluate the performance of our 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 show that our approach can achieve a speedup up to 7.8x with no performance loss compared with the state-of-the-art FPGA solutions for CeNNs.

The remainder of the paper is organized as follows. Section 2 introduces backgrounds and motivation of the paper. The proposed framework for CeNN 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

2.1 Cellular neural networks

Different from the prevalent CNNs which are superior for classification tasks, the CeNN model is inspired by the functionality of visual neurons. In a CeNN, a mass of neuron cells are connected with neighbouring ones, and 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 involve sensory functions, such as noise cancellation, edge detection, path planning, segmentation, etc. For the widely used 2D CeNN with space-invariant templates, the dynamics of each cell state with an M×N rectangular cell array [4] are as follows:

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

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

where $1 \leq i \leq M, 1 \leq j \leq N$, $A_{k,l}(t)$ is the feedback coefficient template, $B_{k,l}(t)$ is the feedforward 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.

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

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

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

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. Please refer to [4] for details. For the mainstream image size with 1920×1080 pixels, the total complexity is 1920×1080×39×100=8.1×10^9 operations with 100 iterations (19 multiplications and 20 additions in each iteration). This warrants exploration of hardware approaches to speedup CeNN computations.

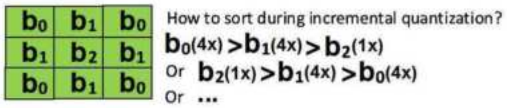

b0 b1 b0
b1 b2 b1
b0 b1 b0
How to sort during incremental quantization?
b0(4x) > b1(4x) > b2(1x)
Or b2(1x) > b1(4x) > b0(4x)
Or ...

Figure 1: CeNN template for binary image noise cancellation application.

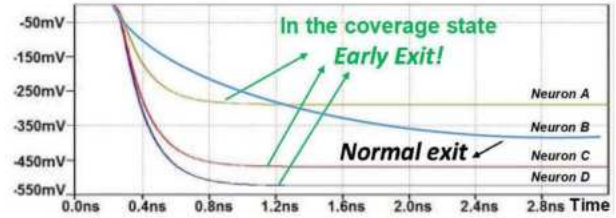


Figure 2: Early exit in CeNN computations.

2.2 Template Learning Algorithm and PSO Algorithm

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

PSO finds solutions (i.e., determining A, B and I templates) 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),

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

where $1 \leq i \leq N$, $1 \leq d \leq D$, 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. The object function for particles taking positions as input is designed according to applications. In CeNN training, PSO will search the space constructed with A, B and I templates, and the templates with the best object function value are obtained as the learned templates.

2.3 Motivation

While hardware oriented memory/computation compression and optimization of CNNs have been extensively studied recently [5][10] [27][21][24][30], 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 CeNN template (template B) for binary image noise cancellation [13] is shown. Only three different values exist for the nine parameters. As such, in [30] 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.

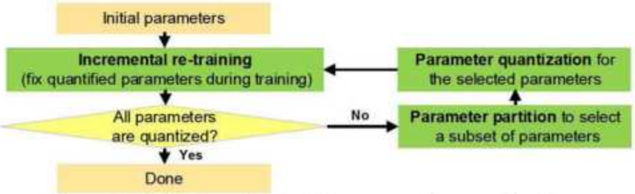


Figure 3: The flowchart of incremental quantization.

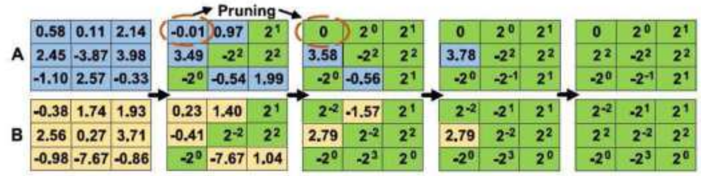


Figure 4: An example of the proposed incremental quantization framework. In each iteration, parameter partition, parameter quantization and incremental re-training are performed sequentially. Green cells represent quantized parameters.

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 5, 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 so as to save energy without sacrificing accuracy.

3. COMPRESSED CENN FRAMEWORK AND HARDWARE IMPLEMENTATION

In this section, we present the compressed CeNN framework including incremental quantization and early exit for CeNN followed by the details of the hardware implementation.

3.1 Incremental Quantization

The proposed incremental quantization framework is an iterative process as shown in Figure 3. Each iteration completes three tasks: parameter partition, parameter quantization, and incremental re-training. We assume 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 4 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 [30], 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. In order 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 towards the end.

Algorithm 1 Weighted nearest neighbor strategy

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

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. 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.

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

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

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.

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

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 = $. In each iteration, a subset of U will be quantized and added to Q.

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

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

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 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 Optimization

Early exit optimization 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 paper, we adopts an experimental method to perform early exit optimization as shown in Fig. 5. 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 [17][28][29], 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

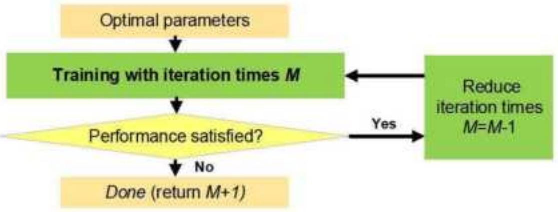


Figure 5: Flowchart of early exit optimization.

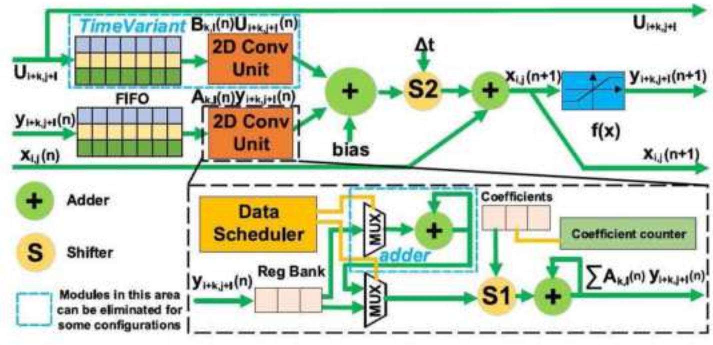


Figure 6: Architecture of the optimized stage design.

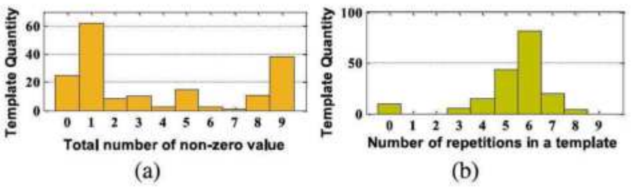
processes the input in a pipelined manner. Furthermore, multiple layers can be connected sequentially for more complex processing or be distributed in parallel for a higher throughput. Note that First In First Out (FIFO) are used between adjacent stages to store the temporary results of each stage (or each iteration), and they are configured as single-input multiple-output memories. Please refer to FPGA implementations in [17][28] for more details.

Our efficient hardware implementation focuses on the optimization of the stage design as shown in Figure 6. 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 Section 3.2.1. 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 3.2.2.

The optimized stage can be configured for both time-invariant templates and time-variant templates. Note that the FPGA implementation [28] is dedicated to CeNN with time-invariant templates, while [17] is for time-variant. The $TimeVariant$ part in Figure 6 is specific for time-variant templates, and can be eliminated in the configuration for time-invariant ones.

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 (Mult.) using LEs and registers.

MODULE	$S1(0)$	$S1(1)$	$S1(2)$	$S1(3)$	$S1(4)$	$S1(5)$	$S2(7)$	$Mult.$
LEs	39	44	50	80	109	105	80	676
REGISTERS	39	42	45	47	50	52	75	486



(a) (b)

Figure 7: Illustration of (a) sparsity and (b) repetition characteristic with 174 CeNN templates.

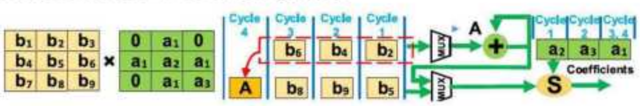


Figure 8: Illustration of sparsity-induced and repetition-induced optimizations.

3.3.1 Shifter Module

In Figure 6, 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 paper is designed to support Δt=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 [8].

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 [11], and totally 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 7(a). In Figure 7(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.

Figure 7(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 8. 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

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

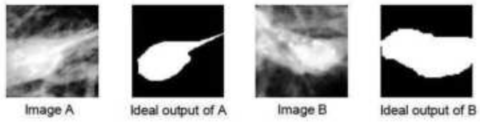


Figure 9: Two selected images and their manually segmented images taken from MIAS database to train CeNN.

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, 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 various incremental quantization strategies and early exit optimization discussed in Section 3 for medical image segmentation. Then we implement the quantized CeNNs on FPGAs and compare their speed with state-of-the-art works.

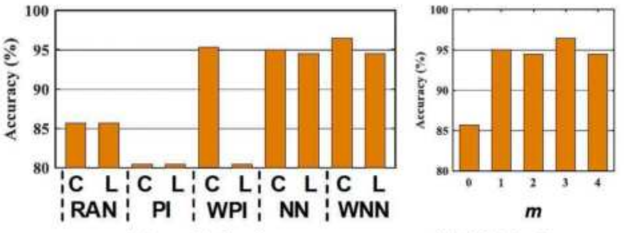
4.1 Performance Evaluation

4.1.1 Experimental Setup

For incremental quantization, 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 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.

The objective function for medical image segmentation in PSO re-training is shown in Equation 12, where *output* and *IdealOutput* 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 [25], and two images and its corresponding segmented results are selected as training images as shown in Figure 9, which is the of the same configuration with the work [23]. 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 as the ideal outputs.



(a) m=2, k=-2

(b) NN-L, k=-m

Figure 10: Performance comparison between templates with various (a) strategies and (b) quantization sizes m for binary image noise cancellation.

$$obj = accuracy = \sum_{i=1}^{t} abs(output_i - IdealOutput_i)/area. \tag{12}$$

4.1.2 Results and Discussion

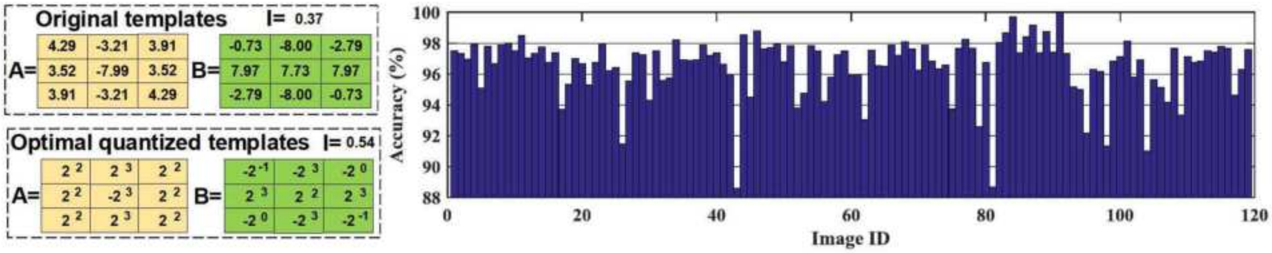
We fix the quantization size using $m = 2$ and $k = -m$, and evaluate all 10 incremental quantization frameworks. The results are depicted in Figure 10(a). From the figure we can observe that the quantized templates achieve similar accuracy compared with the original template without quantization. The 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. Note that generally PI strategy achieves the best performance for CNNs [30]. However, WNN strategy obtains the best performance for CeNN, and NN strategy also obtains a comparable performance. Furthermore, NN and WNN strategy are much stable than PI as NN and WNN can achieve almost the same accuracy for constant and logscale batch sizes while PI not. Even random strategy can have a better accuracy than PI in some configurations. In terms of batch size, constant seems to perform better than log-scale in most cases. It can be interesting in the future to study this in more detail and figure out a systematic way to decide the optimal strategy.

The optimal templates and the original templates are shown in Figure 11, 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 10(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 which gives the best performance and m=3 for medical image segmentation. Further increasing m will not provide any performance gain.

For evaluation of early exit optimization, WNN-C and WPI-L are selected as representatives as shown in Figure 12. It can be noted that the accuracy increases very quickly in the first several iterations, and remains almost constant when the iteration times is lager than a threshold. Usually such thresholds can be a very good candidate as the early exit point. As shown in Figure 12, given the thresholds for early exit optimization, speedups of 2x and 5x are achieved with only 0.8% and 0.9% accuracy loss for WPI-L and WNN-C, respectively.

4.2 Speed Evaluation Using FPGAs

In previous section we have evaluated the performance of our compressed CeNN framework in terms of accuracy. In this section we will evaluate its speed when implemented in FPGAs. For



Original templates I= 0.37

$$A=\begin{bmatrix} 4.29 & -3.21 & 3.91 \\ 3.52 & -7.99 & 3.52 \\ 3.91 & -3.21 & 4.29 \end{bmatrix} \quad B=\begin{bmatrix} -0.73 & -8.00 & -2.79 \\ 7.97 & 7.73 & 7.97 \\ -2.79 & -8.00 & -0.73 \end{bmatrix}$$

Optimal quantized templates I= 0.54

$$A=\begin{bmatrix} 2^{2} & 2^{3} & 2^{2} \\ 2^{2} & -2^{3} & 2^{2} \\ 2^{2} & 2^{3} & 2^{2} \end{bmatrix} \quad B=\begin{bmatrix} -2^{-1} & -2^{3} & -2^{0} \\ 2^{3} & 2^{2} & 2^{3} \\ -2^{0} & -2^{3} & -2^{-1} \end{bmatrix}$$

Figure 11: Performance comparison between the optimal quantized templates and the original templates for medical image segmentation.

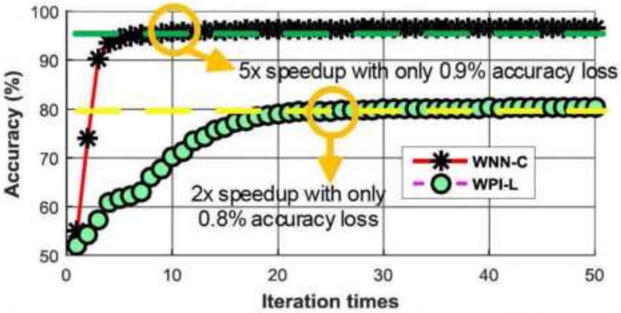


Figure 12: Performance of early exit optimization with WNN-C and WPI-L strategies.

a fair comparison with existing works [17][28][29], 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.

Three configurations of 2D convolution are discussed: one, three and nine multipliers. In Table 3, applying our quantization framework can lead to a 1.2x speedup with increased use of LEs (by 17%) and registers (by 8%) This allows an additional 4 stages to be placed, with a speedup of 1.2x.

Further taking sparsity-induced optimization into consideration, a speedup of 1.8x 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 1.4x 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 [29] and ours with one multiplier (Mult.)/shifter (Shif.) in 2D convolution module, with sparsity-induced optimization and repetition-induced optimization. The numbers in the brackets are the resource utilization rate.

IMPLEMENTATION	STATE-OF -THE-ART (1 MULT.)	OURS (1 SHIF.)	OURS (1 SHIF.+ SPARSITY)	OURS (1 SHIF.+ 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 MULT.	48(100%)	48(100%)	48(100%)	48(100%)
CLOCK F. (MHZ)	353	331	331	322
CYCLES PER PIXEL	11	11	11	8
SPEEDUP	1	**1.2x**	**1.2x**	**1.4x**

Table 4: Speed and resource utilization comparisons of the state-of-the-art work [29] and ours with three and nine multipliers(Mult.)/shifter (Shif.) in 2D convolution module. The numbers in the brackets are the resource utilization rate.

IMPLEMENTATION	STATE-OF -THE-ART (3 MULT.)	OURS (3 SHIF.)	STATE-OF -THE-ART (9 MULT.)	OURS (9 SHIF.)
# 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 MULT.	48(100%)	48(100%)	46(95%)	48(100%)
CLOCK F.(MHZ)	337	320	361	343
CYCLES PER PIXEL	5	5	1	1
SPEEDUP	1	**2.6x**	1	**3.5x**

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 the state-of-the-art work [29] 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 2.6x and 3.5x.

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

Table 5: Speed and resource utilization projections to high-end FPGAs of the state-of-the-art work [29] and ours with nine multipliers/shifters in 2D convolution module. The numbers in the brackets are the resource utilization rate.

Implementation	VC7VX-980T	VC7VX-585T	Stratix V E	Stratix V GS
# of stages	352	179	233	291
LEs($\times10^3$)	780(80%)	465(80%)	718(80%)	524(80%)
Registers($\times10^3$)	170(17%)	93(16%)	133(15%)	128(19%)
Embedded Mult.	3600(100%)	1260(100%)	704(100%)	3926(100%)
Speedup	**2.3x**	**3.3x**	**7.8x**	**1.7x**

1.7x-7.8x. Note that the resource consumption of LEs and registers are almost the same for all the implementations, and the speedup varies with the number of embedded multipliers, or more specifically, the ratio of LEs to embedded multipliers. A high ratio of LEs to embedded multipliers means more LEs can be used to implement shifters resulting with a high speedup. The highest speedup of 7.8x is due to the fact that the Stratix V E FPGA has the highest rate of LEs to embedded multipliers.

5. CONCLUSIONS

In this paper, we propose a compressed CeNN framework for computation reduction in CeNNs for edge segmentation. Particularly, we present powers-of-two based incremental quantization and early exit optimization. The incremental quantization adopts an iterative procedure including parameter partition, parameter quantization, and re-training 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 optimization is presented with an experimental method, and the configuration and performance is determined by specific applications Experimental results on medical image segmentation show that the proposed framework can achieve similar performance compared with that using original templates without optimization, and the implementation with incremental quantization can achieve a speedup up to 7.8x compared with the state-of-the-art FPGA implementations, while early exit optimization can obtain a speedup of 5x for medical image segmentation. We also discover that unlike CNNs, the optimal strategy of CeNNs is weighted nearest neighbor strategy other than pruning-inspired strategy.

References

[1] S. J. Carey, D. R. Barr, B. Wang, A. Lopich, and P. Dudek. Mixed signal simd processor array vision chip for real-time image processing. *Analog Integrated Circuits and Signal Processing*, 77(3):385–399, 2013.

[2] S.-H. Chae, D. Moon, D. G. Lee, and S. B. Pan. Medical image segmentation for mobile electronic patient charts using numerical modeling of iot. *Journal of Applied Mathematics*, 2014, 2014.

[3] H.-C. Chen, Y.-C. Hung, C.-K. Chen, T.-L. Liao, and C.-K. Chen. Image-processing algorithms realized by discrete-time cellular neural networks and their circuit implementations. *Chaos, Solitons & Fractals*, 29(5):1100–1108, 2006.

[4] L. O. Chua and T. Roska. *Cellular neural networks and visual computing: foundations and applications*. Cambridge university press, 2002.

[5] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[6] F. Dohler, F. Mormann, B. Weber, C. E. Elger, and K. Lehnertz. A cellular neural network based method for classification of magnetic resonance images: towards an automated detection of hippocampal sclerosis. *Journal of neuroscience methods*, 170(2):324–331, 2008.

[7] M. Duraisamy and F. M. M. Jane. Cellular neural network based medical image segmentation using artificial bee colony algorithm. In *Green Computing Communication and Electrical Engineering (ICGCCEE), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[8] H. Harrer and J. A. Nossek. Discrete-time cellular neural networks. *International Journal of Circuit Theory and Applications*, 20(5):453–467, 1992.

[9] H. Harrer, J. A. Nossek, T. Roska, and L. O. Chua. A current-mode dtcnn universal chip. In *Circuits and Systems, 1994. ISCAS'94., 1994 IEEE International Symposium on*, volume 4, pages 135–138. IEEE, 1994.

[10] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, pages 4107–4115, 2016.

[11] K. Karacs, G. Cserey, Zarndy, P. Szolgay, C. Rekeczky, L. Kek, V. Szab, G. Pazienza, and T. Roska. Software library for cellular wave computing engines. *Cellular Sensory and Wave Computing Laboratory of the Computer and Automation Research Institute*, 2010.

[12] S. Lee, M. Kim, K. Kim, J.-Y. Kim, and H.-J. Yoo. 24-gops 4.5-mm^2 digital cellular neural network for rapid visual attention in an object-recognition soc. *IEEE transactions on neural networks*, 22(1):64–73, 2011.

[13] H. Li, X. Liao, C. Li, H. Huang, and C. Li. Edge detection of noisy images based on cellular neural networks. *Communications in Nonlinear Science and Numerical Simulation*, 16(9):3746–3759, 2011.

[14] M. Magdalena and U. G. B. Bujnowska-Fedak. Use of telemedicine-based care for the aging and elderly: promises and pitfalls. *Smart homecare Technology & telehealth*, 3:91–105, 2015.

[15] D. Manatunga, H. Kim, and S. Mukhopadhyay. Sp-cnn: A scalable and programmable cnn-based accelerator. *IEEE Micro*, 35(5):42–50, 2015.

[16] G. Manganaro, P. Arena, and L. Fortuna. *Cellular neural networks: chaos, complexity and VLSI processing*, volume 1. Springer Science & Business Media, 2012.

[17] J. J. Martnez, J. Garrigs, J. Toledo, and J. M. Ferrndez. An efficient and expandable hardware implementation of multilayer cellular neural networks. *Neurocomputing*, 114:54–62, 2013.

[18] J. Muller, R. Wittig, J. Muller, and R. Tetzlaff. An improved cellular nonlinear network architecture for binary and greyscale image processing. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2016.

[19] R. Porter, J. Frigo, A. Conti, N. Harvey, G. Kenyon, and M. Gokhale. A reconfigurable computing framework for multi-scale cellular image processing. *Microprocessors and Microsystems*, 31(8):546–563, 2007.

[20] S. Potluri, A. Fasih, L. K. Vutukuru, F. Al Machot, and K. Kyamakya. Cnn based high performance computing for real time image processing on gpu. In *Nonlinear Dynamics and Synchronization (INDS) & 16th Int'l Symposium on Theoretical Electrical Engineering (ISTET), 2011 Joint 3rd Int'l Workshop on*, pages 1–7. IEEE, 2011.

[21] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[22] A. Rodrguez-Vzquez, G. Lin-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galn, F. Jimnez-Garrido, R. Domnguez-Castro, and S. E. Meana. Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(5):851–863, 2004.

[23] R. Rouhi, M. Jafari, S. Kasaei, and P. Keshavarzian. Benign and malignant breast tumors classification based on region growing and cnn segmentation. *Expert Systems with Applications*, 42(3):990–1002, 2015.

[24] H. Song, P. Jeff, T. John, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations*, 2016.

[25] J. Suckling, J. Parker, D. Dance, S. Astley, I. Hutt, C. Boggis, I. Ricketts, E. Stamatakis, N. Cerneaz, S. Kok, et al. The mammographic image analysis society digital mammogram database. In *Exerpta Medica. International Congress Series*, volume 1069, pages 375–378, 1994.

[26] H. Wong, V. Betz, and J. Rose. 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*, pages 5–14. ACM, 2011.

[27] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[28] N. Yildiz, E. Cesur, K. Kayaer, V. Tavsanoglu, and M. Alpay. Architecture of a fully pipelined real-time cellular neural network emulator. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(1):130–138, 2015.

[29] N. Yildiz, E. Cesur, and V. Tavsanoglu. On the way to a third generation real-time cellular neural network processor. *CNNA 2016*, 2016.

[30] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *5th International Conference on Learning Representations*, 2017.