A Mixed Signal Architecture for Convolutional Neural Networks

QIUWEN LOU, University of Notre Dame
CHENYUN PAN, University of Kensas
JOHN MCGUINNESS, University of Notre Dame
ANDRAS HORVATH, Pazmany Peter Catholic University
AZAD NAEEMI, Georgia Institute of Technology
MICHAEL NIEMIER and X. SHARON HU, University of Notre Dame

Deep neural network (DNN) accelerators with improved energy and delay are desirable for meeting the requirements of hardware targeted for IoT and edge computing systems. Convolutional neural networks (CoNNs) belong to one of the most popular types of DNN architectures. This article presents the design and evaluation of an accelerator for CoNNs. The system-level architecture is based on mixed-signal, cellular neural networks (CeNNs). Specifically, we present (i) the implementation of different layers, including convolution, ReLU, and pooling, in a CoNN using CeNN, (ii) modified CoNN structures with CeNN-friendly layers to reduce computational overheads typically associated with a CoNN, (iii) a mixed-signal CeNN architecture that performs CoNN computations in the analog and mixed signal domain, and (iv) design space exploration that identifies what CeNN-based algorithm and architectural features fare best compared to existing algorithms and architectures when evaluated over common datasets—MNIST and CIFAR-10. Notably, the proposed approach can lead to 8.7× improvements in energy-delay product (EDP) per digit classification for the MNIST dataset at iso-accuracy when compared with the state-of-the-art DNN engine, while our approach could offer 4.3× improvements in EDP when compared to other network implementations for the CIFAR-10 dataset.

CCS Concepts: • **Hardware → Application specific integrated circuits**; *Application specific processors*;

Additional Key Words and Phrases: Hardware accelerator, convolutional neural networks, analog circuits

ACM Reference format:
Qiuwen Lou, Chenyun Pan, John McGuinness, Andras Horvath, Azad Naeemi, Michael Niemier, and X. Sharon Hu. 2019. A Mixed Signal Architecture for Convolutional Neural Networks. *J. Emerg. Technol. Comput. Syst.* 15, 2, Article 19 (March 2019), 26 pages.
https://doi.org/10.1145/3304110

This work was supported in part by the Center for Low Energy Systems Technology (LEAST), one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This project was also supported by the National Science Foundation under grant 1640081, and the Nanoelectronics Research Corporation (NERC), a wholly-owned subsidiary of the Semiconductor Research Corporation (SRC), through Extremely Energy Efficient Collective Electronics (EXCEL), an SRC-NRI Nanoelectronics Research Initiative under Research Task ID2698.004.
Authors' addresses: Q. Lou, J. McGuinness, M. Niemier, and X. S. Hu, 100 Notre Dame Avenue, Notre Dame, IN, 46637, USA; C. Pan, University of Kensas, Lawrence, Kensas, USA; A. Horvath, Pazmany Peter Catholic University, Szentkiralyi U. 28, 1008, Budapest, Hungary; A. Naeemi, Georgia Institute of Technology, Atlanta, Georgia, USA.

1550-4832/2019/03-ART19 $15.00
https://doi.org/10.1145/3304110

1 INTRODUCTION

In the machine-learning community, there is great interest in developing computational models to solve problems related to computer vision [32], speech recognition [16], information security [25], climate modeling [23], and so on. To improve the *delay and energy efficiency* of computational tasks related to both inference and training, the hardware design and architecture communities are considering how hardware can best be employed to realize algorithms/models from the machine-learning community. Approaches include application specific circuits (ASICs) to accelerate deep neural networks (DNNs) [50, 59] and convolutional neural networks (CoNNs) [41], neural processing units (NPUs) [18], hardware realizations of spiking neural networks [14, 28], and so on.

When considering application-specific hardware to support neural networks, it is important that said hardware can implement networks that can be extensible to a large class of networks and solve a large collection of application-level problems. Deep neural networks (DNNs) represent a class of such networks and have demonstrated their strength in applications such as playing the game of Go [54], image and video analysis [32], target tracking [31], and so on. In this article, we use convolutional neural network (CoNN) as a case study for DNNs due to its general prevalence. CoNNs are computationally intensive, which could lead to high latency and energy for inference and even higher latency/energy for training. The focus of this article is on developing a low energy/delay mixed-signal system based on cellular neural networks (CeNNs) for realizing CoNN.

A Cellular Nonlinear/Neural Network (CeNN) is an analog computing architecture [11] that could be well suited for many information processing tasks. In a CeNN, identical processing units (called cells) process analog information in a concurrent manner. Interconnection between cells is typically local (i.e., nearest neighbor) and space invariant. For spatio-temporal applications, CeNNs can offer vastly superior performance and power efficiency when compared to conventional von Neumann architectures [47, 61]. Using "CeNNs for CoNN" allows the bulk of the computation associated with a CoNN to be performed in the analog domain. Sensed information could immediately be processed with no analog-to-digital conversion (ADC). Also, inference-based processing tasks can tolerate lower precision (e.g., Google's TPU employs 8-bit integer matrix multiplies [24]) typically associated with analog hardware and can leverage its higher energy efficiency. With this context, we have made the following contributions in this article.

(i) We elaborate the use of CeNN to realize computations that are typically associated with different layers in a CoNN. These layers include convolution, ReLU, and pooling. Based on the implementations for each layer, a baseline CeNN-friendly CoNN for the MNIST problem [36] is presented.[1]
(ii) We introduce an improved CoNN model for the MNIST problem to support CeNN-friendly layers/algorithms that could ultimately improve figures of merit (FOM) such as delay, energy, and accuracy, and so on. Following the same concept, we also develop a CeNN-friendly CoNN for the CIFAR-10 problem [33].
(iii) We present a complete, mixed-signal architecture to support CeNN-friendly CoNN designs. Besides CeNN cells and SRAM to store weights, the architecture includes analog memory to store intermediate feature map data and ADC and digital circuits for the FC layer computation. The architecture also supports efficient programming/reprogramming CeNN cells.

We have conducted detail studies of energy, delay, and accuracy per classification for the MNIST and CIFAR-10 datasets and compared our networks and architecture with other algorithms and

[1] A preliminary version of the design was presented in Reference [19].

architectures [14, 18, 28, 41, 50, 59] that address the same problem. For the MNIST dataset, at iso-accuracy, our results demonstrate an 8.7× improvement in energy-delay product (EDP) when compared with a state-of-the-art accelerator. When compared with another recent analog implementation [5], a 10.3× improvement in EDP is observed. For the CIFAR-10 dataset, a 4.3× improvement in EDP is observed when comparing with a state-of-the-art quantized approach [18].

The rest of the article is structured as follows. Section 2 gives a general discussion of CeNNs and existing CoNN accelerators. In Section 3, we present the implementation of CoNN layers in CeNNs. Our baseline network designs as well as other algorithmic changes and network topologies that might be well suited for our architecture are given in Section 4. Section 5 describes our proposed architecture, including CeNN cell design, and simulations of various core architectural components. Evaluation and benchmarking results are presented in Section 6. Last, Section 7 concludes the article.

2 BACKGROUND

Here, we briefly review the basic concepts of CeNN and accelerator designs for CoNN.

2.1 CeNN Basics

A CeNN architecture is a spatially invariant, $M \times N$ array of identical cells (Figure 1(a)) [19]. Each cell C_{ij} has identical connections with adjacent cells in a predefined neighborhood. These neighborhood cells are denoted as $N_r(i, j)$ of radius r (i.e., a given cell communicates with other cells within a neighborhood r). The number of cells (m) in the neighborhood is given by the expression $m = (2r + 1)^2$. (r is typically 1, which suggests that each cell interacts with only its immediate neighbors.)

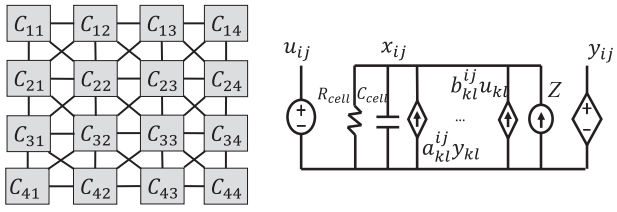
A CeNN cell is composed of one resistor, one capacitor, $2m$ linear voltage-controlled current sources (VCCSs), and one fixed current source (Figure 1(b)). A cell's input, state, and the output of a given cell, C_{ij}, correspond to the nodal voltages, u_{ij}, x_{ij}, and y_{ij}, respectively. VCCSs controlled by input and output voltages of each neighbor deliver feedforward and feedback currents to a given cell. To understand CeNN cell dynamics, we can simply assume a system of $M \times N$ ordinary differential equations. Each equation is simply the Kirchhoff's Current Law (KCL) at the state nodes of the corresponding cells (Equation (1)). CeNN cells also employ a non-linear sigmoid-like transfer function at the output (see Equation (2)),

$$C_{cell}\frac{dx_{ij}(t)}{dt} = -\frac{x_{ij}(t)}{R_{cell}} + \sum_{C_{kl}\in N_r(i,j)} a_{ij,kl}y_{kl}(t) + \sum_{C_{kl}\in N_r(i,j)} b_{ij,kl}u_{kl} + Z, \tag{1}$$

$$y_{k,l} = \frac{1}{2}\left|x_{k,l}+1\right| - \frac{1}{2}\left|x_{k,l}-1\right|. \tag{2}$$

Feedback and feed-forward weights from cell C_{kl} to cell C_{ij} are captured by the parameters $a_{ij,kl}$ and $b_{ij,kl}$, respectively. $a_{ij,kl}$, and $b_{ij,kl}$ are space invariant and are denoted by two $(2r + 1) \times (2r + 1)$ matrices. (If $r = 1$, then matrices are 3×3.) Matrices of a and b parameters are referred to as templates—where A and B are the feedback and feed-forward templates, respectively. Template values are the coefficients in the differential equation and can either be a constant to reflect a linear relationship between cells or a non-linear function (which can be dependent on the input or state of the corresponding neighboring cell) to reflect non-linear relationship between cells. Design flexibility is further enhanced by the fixed bias current Z. This provides a means to adjust total current flowing into a cell. By selecting values for A, B, and Z, CeNNs can solve a wide range of problems.

Various circuits including inverters, Gilbert multipliers, operational transconductance amplifiers (OTAs), and so on can be used as VCCSs in CeNN [22, 37]. For the work to be discussed in



(a) (b)

Fig. 1. (a) CeNN array architecture; (b) CeNN cell circuitry.

this article, we assume the OTA design from Reference [40]. OTAs provide a large linear range for voltage to current conversion and can implement a wide range of transconductances that could be used for different CeNN template implementations. Furthermore, these OTAs can also be used to implement *Non-linear* templates, which leads to CeNNs with richer functionality [40].

2.2 Convolutional Neural Network Accelerators

Due to the high computational complexity of CoNNs, various hardware platforms are used to enable the efficient processing of DNNs, including GPUs, FPGAs, ASICs, and so on. Specifically, there is a growing interest in using ASICs to provide more specialized acceleration of DNN computation. A recent review paper summarized these approaches in Reference [56]. Both digital and/or analog circuitries are proposed to implement these accelerators. In the digital domain, typical approaches include using optimized dataflow to efficiently reduce the data movement overhead for the dense matrix multiplication operation [8] or implementing sparse matrix multiplication by applying pruning to the network [17]. Recently, analog implementations have also been proposed to accelerate deep learning processes. Work in Reference [5] embedded a charge sharing scheme into SRAM cells to reduce the overhead of memory accesses. Work in Reference [53] uses a crossbar circuit with memristors to speed up the inference of deep neural networks.

3 CENN IMPLEMENTATION OF CONN COMPUTATIONS

As pointed out earlier, CeNNs have a number of benefits such as (i) ease of implementation in VLSI, (ii) low energy due to its nature fit for analog realization, (iii) Turing complete, and so on. We show in this section that all the core operations in a CoNN can be readily implemented with CeNNs. In a CoNN, every layer typically implements a simple operation that might include (i) convolutions, (ii) non-linear operations (usually a rectifier), (iii) pooling operations, and (iv) fully connected layers. Below we describe how each of these layers can map to a CeNN. A more detailed description of the operations and how the layered network itself can be built can be found in References [15, 34]. We will also discuss our network design in Section 4.

3.1 Convolution

Convolution layers are used to detect and extract different feature maps on input data as the summation of the pointwise multiplication of the feature map and the convolutional kernel. One map is the input image (f), and the convolutional kernel encodes a desired feature (g) to be detected by some operation. It is easy to see that a convolution has the highest response at positions where the desired feature appears. The convolution operation can be defined per Equation (3). The exact convolutional kernels are optimized during training,

$$f * g(i,j) = \sum_{k,l=-\infty}^{\infty} f(i-k, j-l)g(k,l). \tag{3}$$

As can be seen from Equation (1), with the application of the feed-forward template (denoted as $b_{ij,kl}$), one CeNN can implement a convolutional kernel for a feature map in a straightforward manner. Then, all these feature maps after convolutional operations need to sum up together. We will discuss the mechanism for achieving this in Section 5.

Due to the sigmoid function within the CeNN equation, the output of CeNN is thresholded to the range $(-1, 1)$. However, in the CoNN computation, the output could be larger than 1 or less than -1, which leads to an error in data representation. However, our initial simulation results suggest that this error does not impact the overall classification accuracy in the networks considered in this article.

3.2 Rectified Linear Units

As CoNNs are built and designed for recognition purposes and classification tasks, non-linear operations are required. Perhaps the most commonly used non-linearity in deep learning architectures [12] is the rectified linear unit (ReLU) that per Equation (4), thresholds every value below zero,

$$R(x) = \left\{ \begin{matrix} 0, if x \leq 0 \\ x, if x > 0 \end{matrix} \right\}. \tag{4}$$

In a CeNN, the ReLU operation can be implemented using a non-linear template. In CeNN theory, nonlinear templates are usually noted as $\hat{D}$ templates in parallel with A templates and B templates. To realize the required non-linear computation here, one can define an additional template implementing the non-linear function of the ReLU operation: $\hat{D}(x_{i,j}) = \max(0, x_{i,j})$. This function sets all negative values to zero and leaves the positive values unchanged, and hence it directly implements Equation (4). That said, (i) while non-linear templates are well established in the *theory* of CeNNs, (ii) the application of a non-linear function has obvious computational utility, and (iii) non-linear templates can be easily simulated, in practice, non-linear operations are much more difficult to realize. While existing hardware considers non-linear template implementations [40], it may still not exactly mimic the behavior of non-linear templates. (We will discuss this in more detail in Section 3.4.)

Alternatively, as the CeNN-UM is Turing complete, all non-linear templates can be implemented as a series of linear templates together with the implicit CeNN non-linearity (i.e., sigmoid output, see Equation (2)) [52]. This implicit CeNN non-linearity is widely implemented in real devices such as the ACE16k chip [51] or the SPS 02 Smart Photosensor from Toshiba [1]. In the CoNN case, the ReLU operation can be rewritten as a series of linear operations (with the implicit CeNN non-linearity) by applying templates below.

First, one can execute the feed-forward template given by Equation (5), which simply decreases all values by 1. Because the standard CeNN non-linearity thresholds all values in a CeNN array below -1, after this shift all values between -1 and 0 are simply set to -1:

$$B_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, Z = -1. \tag{5}$$

Next, one can shift the values back (i.e., increase them by 1) by applying the template operation in Equation (6):

$$B_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, Z = 1. \tag{6}$$

As the non-linearity thresholds a given value, these two linear operations implement the required ReLU operator, i.e., leaving all positive values unchanged, and thresholds all values below 0.

3.3 Pooling

Pooling operations are employed to decrease the amount of information flow between consecutive layers in a deep neural network to compensate for the effects of small translations. Two pooling approaches are widely used in CoNN: max pooling and average pooling. Here, we discuss the implementations of both pooling approaches using CeNN.

3.3.1 Max Pooling. A max pooling operation selects the maximum element in a region around every value per Equation (7):

$$P(i,j) = \max_{k,l \in S} f(i-k, j-l). \tag{7}$$

Similarly to ReLU, max pooling is also a non-linear function. As before, functionality associated with max pooling can also be realized with a sequence of *linear* operations. We use a pooling operation with a 3×3 receptive field as an example to illustrate the process. The idea here is to compare the intensity of each pixel in the image with all its neighbors in succession (with a radius of 1 in the 3×3 case). We use $x_{i,j}$ to represent the intensity for pixel (i, j). For each comparison, if the intensity of its neighbor pixel (defined as $x_{k,l}$) is larger than $x_{i,j}$, then we use $x_{k,l}$ to replace $x_{i,j}$ in the location (i, j); otherwise, $x_{i,j}$ remains unchanged. By making comparisons with all neighboring pixels, the value of $x_{i,j}$ can be set to the magnitude of all of its neighbors.

We developed a sequence of CeNN templates to realize the comparison between $x_{i,j}$ and all its neighboring pixels, $x_{k,l}$. Then, by simply rotating the templates, we can easily compare $x_{i,j}$ to other neighbor pixels. Downsampling could be performed afterwards to extract the maximum value within a certain range if needed. The detailed CeNN operations to realize the comparison can be broken down into four steps and are summarized as follows. (i) Apply the linear DIFF template shown in Equation (8):

$$B_1 = \begin{bmatrix} 0 & 0.5 & 0 \\ 0 & -0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}, Z = -1. \tag{8}$$

The output after applying this template is $y = -0.5x_{i,j} + 0.5x_{k,l} - 1$. After applying the sigmoid function, $y = -1$ if $x_{i,j} \geq x_{k,l}$; otherwise, y remains unchanged. (ii) Apply the linear INC template in Equation (9) to shift the pixel intensity up. After this operation, y becomes 0 if $x_{i,j} \geq x_{k,l}$; otherwise, $y = -0.5x_{i,j} + 0.5x_{k,l}$,

$$B_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, Z = 1. \tag{9}$$

(iii) Apply the CeNN MULT template to multiply y by 2. Thus, $y = 0$ if $x_{i,j} \geq x_{k,l}$; otherwise, $y = -x_{i,j} + x_{k,l}$. (iv) Add $x_{i,j}$ to y to obtain the maximum between $x_{k,l}$ and $x_{i,j}$, and use it to update the intensity in the location (i, j).

3.3.2 Average Pooling. Per Section 3.3.1, a max pooling operation with linear CeNN templates requires up to 16 computational steps. (Each comparison requires four steps, while the pixel needs to compare with (at least) its neighboring four pixels.) That said, average pooling can be used in lieu of max pooling and may have only a nominal impact on the classification accuracy in certain scenarios [6]. Average pooling operations can be easily realized with CeNNs; in fact, only one

template operation is required. To perform an average pooling operation in 2×2 or 3×3 grids, one can simply employ the B templates in Equation (10) ($Z = 0$),

$$B_{2x2} = \begin{bmatrix} 1/4 & 1/4 & 0 \\ 1/4 & 1/4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B_{3x3} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}. \tag{10}$$

3.4 Non-linear Template Operations

While CeNN templates typically suggest linear relationships between cells, non-linear relationships are also possible and can be highly beneficial. (As noted earlier, while non-linear template operations are well supported by CeNN theory, in hardware realizations, linear operations are more common owing to the complexity of the circuitry required for non-linear steps.) That said, we also consider what impact non-linear template operations may ultimately have on application-level FOM.

We consider non-linear implementations of ReLU and pooling per Reference [40]. The non-linear OTA based I-V characteristic shown in Reference [40] can directly mimic the ReLU function discussed in Section 3.2. The pooling operation can also be implemented by the non-linear, GLOBMAX template, which can be found in the standard CeNN template library [2]. The GLOBMAX operation selects the maximum value in the neighborhood of a cell in a CeNN array and propagates it through the array. By setting the execution time of the template accordingly, one can easily set how far the maximum values can propagate/which regions the maximum values can fill. Here, the non-linear templates can also be implemented by using the $\hat{D}$ type non-linear function as given in Equation (11),

$$\hat{D}(x_{i,j}) = \left\{ \begin{array}{l} -\frac{1}{8}x, \text{ if } x \leq 0 \\ 0, \text{ if } x > 0 \end{array} \right\}. \tag{11}$$

3.5 Fully Connected Layers

The operations described above are used in local feature extractors and can extract complex feature maps from a given input image. However, to accomplish classification, one must convert said feature maps into a scalar index value associated with the selected class. While various machine-learning algorithms (e.g., SVMs) can be used for this, a common approach is to employ a fully connected (FC) layer and associated neurons. The FC layer considers information globally and unifies local features from the lower layers. It can be defined as a pixelwise dot product between a weight map and the feature map. This product can be used as a classification result, which captures how strongly the data belongs to a class and the product is calculated for every class independently. The index of the largest classification result can be selected and associated with the input data.

CeNNs can be readily used to implement the dot product function in the FC layer. However, if for large feature maps and weight maps, i.e., the pointwise calculation for vector length over 9, CeNN would require large $r's$ and hence cannot efficiently implement such FC layers. To overcome this challenge, one can leverage a digital processor (e.g., per Reference [43]) to perform the FC layer function.

4 CENN-BASED CONNS FOR TWO CASE STUDIES

As mentioned in the previous section, (a) CeNNs could operate in the analog domain—which could result in lower power consumption/improved energy efficiency [29], and (b) CeNNs are Turing complete [10] and could provide a richer library of functionality than which is typically associated with CoNNs. In this section, we consider how the topographic, highly parallel CeNN architecture can efficiently implement deep-learning operations/CoNNs.

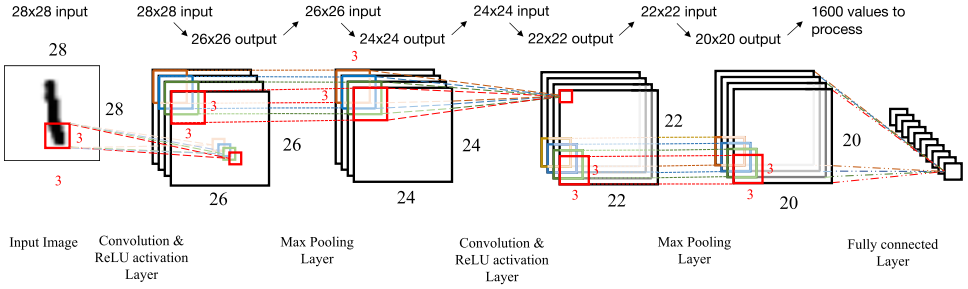


Fig. 2. CeNN-friendly CoNN for the MNIST problem—design 1.

CeNNs are typically composed of a single layer of processing elements (PEs). Thus, while most CeNN hardware implementations lack the layered structure of CoNNs, by using local memory and CeNN reprogramming (commonly available on every realized CeNN chip [51] as will be discussed), a cascade of said operations can be realized by re-using the result of each previous processing layer [10]. One could also simply use multiple CeNNs to compute different feature maps in each layer in parallel. These CeNNs need to communicate with each other, e.g., to sum values for different feature maps. Below, we show how the layered CoNNs can be realized with layers of CeNNs through two case studies: (i) MNIST and (ii) CIFAR-10.

4.1 CeNN-based CoNNs for MNIST

Using the building blocks described above, we have developed several CeNN-friendly structures for the MNIST problem. In the MNIST handwritten digit classification task [35], a system must analyze and classify what digit (0–9) is represented by a 28×28 pixel gray scale image. There are 60,000 images in the training set and 10,000 images in the test set.

To develop the CeNN-friendly CoNN, we leverage the following two observations. First, all computational kernels are best to be restricted to a CeNN friendly size of 3×3. In some sense, this could be viewed as a "departure" from larger kernel sizes (e.g., 7×7 or larger) that may be common in CoNNs. It should be noted that larger kernels are acceptable according to the CeNN theory (i.e., per Section 2, a neighborhood's radius r could easily be larger than 1). However, due to increased connectivity requirements, said kernels are infrequently realized in hardware. That said, the 3×3 kernel size is not necessarily a restriction. Recent works [55] suggests that larger kernels can be estimated by using a series of 3×3 kernels with fewer parameters. Again, this maps well to CeNN hardware. Second, per the discussion in Section 3, all template operations for the convolution, ReLU, and pooling steps are feed-forward (B) templates. The feedback template (A) is not used in any of the feature extracting operations (i.e., per Equation (1), all values would simply be 0).

During network development, we use TensorFlow to train the network with full precision to obtain accuracy data. We use stochastic gradient descent for training, with the initial learning rate set to 10^{-2}. We have also implemented a more versatile/adjustable training framework in MATLAB. The MATLAB based simulator extracts weights from the trained model (from TensorFlow), and performs inference in conjunction with CeNN operations at the precision that is equivalent to actual hardware. Our network learns the parameters of the B-type templates for the convolution kernels. (Per Section 3, the B-template values for the ReLU and pooling layers are fixed.)

Following the observations and process described above, we develop a layered, CeNN-friendly network to solve the MNIST problem. The network topology is shown in Figure 2. The network contains two convolution layers, and each layer contains four feature maps. There is also an FC

Table 1. Classification Accuracy for Different CoNN Designs for the MNIST Problem

Approach	Network in Figure 2	Network in Figure 3
Baseline	98.1%	97.8%
Average pooling	97.5%	96.7%
Nonlinear templates	93.1%	91.5%

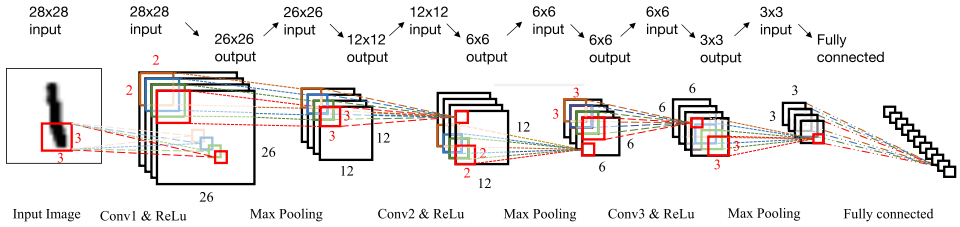


Fig. 3. CeNN-friendly CoNN for the MNIST problem—design 2.

layer that follows the two convolution layers to obtain the classification results. The baseline network is designed using maximum pooling and linear templates to potentially maximize the classification accuracy. However, we also study the network accuracy for average pooling and alternatives based on non-linear templates to evaluate tradeoffs in terms of accuracy, delay, energy, and so on, to be discussed.

The accuracy for different design options for the network are summarized in the second column in Table 1. From the table, we can see that max pooling generally leads to better accuracy than average pooling. The non-linear template implementation is also less accurate than the linear implementation for max pooling. This is mainly because the GLOBALMAX template is an approximation for the max pooling, and it is *not* as accurate as the linear template approach.

4.2 Eliminating FC Layers

One of the potential challenges of a network with a fully connected layer shown in Figure 2 is the need to convert analog CeNN data into a digital representation to perform computations associated with an FC layer, since an FC layer is not CeNN friendly (see Section 3.5). To reduce the impact of analog-to-digital conversion and associated FC layer computation, we have designed an alternative network for MNIST digit classification to perform computations associated with an FC layer.

In this alternative network (Figure 3), the weights (and image sizes) associated with the last layer of the network are reduced to CeNN-friendly, 3×3 kernels. Changes include modifications to the pooling layer. In the network in Figure 2, max pooling is achieved by propagating the maximum pixel value to all neighbors within a certain region specified by the network design. However, the sizes of these feature maps do not change. For the network in Figure 3, the maximum value is propagated within a 2×2 grid to form a group, and only one maximum pixel value in each group is extracted to be processed in the next stage of the network. Thus, the network size is reduced by a factor of two with each pooling layer. For the implementation of downsampling through max pooling, after a pooling operation is completed, for each a 2×2 grid within a feature map, only one pixel is required to write to an analog memory array for the next stage processing. In the network in Figure 3, three pooling layers are required to properly downsize an image and obtain reasonable accuracy. The final computational steps associated with this alternative network are

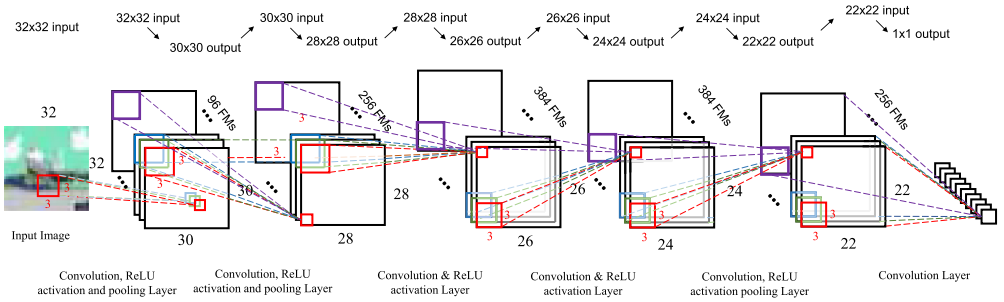


Fig. 4. CeNN-friendly CoNN for CIFAR-10 problem.

readily amenable for CeNN hardware implementations. However, both the image size and the kernel size are reduced to 3×3.

Potential overheads associated with FC layer computations are reduced as only the final results (10 probability values corresponding to the number of image classes) must be sent to any digital logic and/or CPU (in lieu of the 16,000 signals associated with the network in Figure 2). Downsampling may also impact classification energy, as smaller subsets of the CeNN array can be used for computations associated with successive layers in the network. Again, we evaluate the accuracy of this proposed approach by using average pooling, nonlinear templates, and so on. The results are shown in the third column in Table 1. In general, these accuracy numbers are still close to the baseline design discussed in Section 4.1.

In general, this strategy should be applicable to any network, regardless of its depth and width and the kernel sizes employed. By properly downsampling the feature map in the relevant layer (i.e., to reduce the feature map size by 1/2 or 1/3 when needed), we can eventually obtain a 3×3 feature map in the last layer of a given network.

4.3 CeNN-based CoNNs for CIFAR-10

The networks proposed in Section 4.1 and Section 4.2 for MNIST are relatively simple compared with state-of-the-art networks. Typically, to solve more complex problem, larger networks with more layers/feature maps are required. In this subsection, we discuss our design for larger CeNN-friendly CoNNs.

As a case study, we use CIFAR-10 as the dataset, which consists of 50,000 images in the training set, 10,000 images in the validation set and 10,000 images in the test set. These images are all color images with RGB channel. There are 10 classes with different objects (e.g., airplane, automobile, bird, etc.) within the dataset. Each image belongs to one class, with a size of 32×32. During the inference stage, the network must predict which class the image belongs to.

We use modified AlexNet [32] network to solve the CIFAR-10 problem. AlexNet is originally used to solve ImageNet [13], which is a more complex problem. Thus, we expect our modification still leads to reasonable accuracy for CIFAR-10. We perform our modifications on AlexNet to (i) enable the modified network to solve the CIFAR-10 problem and (ii) make the network CeNN-friendly. Specifically, our main modifications are summarized as follows: (i) For all convolution layers in AlexNet, the kernel sizes are changed to 3×3 so that it is readily amendable to CeNNs with the same template size. (ii) We remove the FC layer in the AlexNet, since it is not CeNN-friendly, and use a convolution layer with 10 outputs as the last layer to obtain the classification probabilities. (iii) Downsampling in the pooling layer is not used in the modified network to retain reasonable model size. The network architecture is shown in Figure 4.

Table 2. Classification Accuracy for Different CoNN Designs for the CIFAR-10 Problem

Approach	CeNN-friendly AlexNet C96-C256-C384-C384-C256	CeNN-friendly AlexNet C64-C128-C256-C256-C128	CeNN-friendly AlexNet C64-C128-C128-C128-C64
Accuracy	84.5%	82.9%	81.8%

We use the network in Figure 4 as a baseline and explore the design space by (1) changing the number of feature maps in each layer, (2) using the downsampling approach mentioned in Section 4.2.

In the baseline, the feature maps for the first five convolution layers are the same as AlexNet (C96-C256-C384-C384-C256). We also considered feature map sizes of C64-C128-C256-C256-C128 and C64-C128-C128-C128-C64. We use the Adam algorithm [30] to train the network, with learning rate set to 10^{-4}. The accuracy data for different design options are summarized in Table 2. The accuracies only drop for 1.6% and 2.17% with the decrease of the network size. Therefore, we also consider these two networks in the benchmarking efforts discussed in Section 6.

We also use the approach mentioned in Section 4.2 to resize the feature maps of selective layers, to make the size of each feature map in last layer 3×3. The feature maps of the five layers in the CeNN-friendly AlexNet become 32×32−>16×16−>8×8−>4×4−>3×3, which makes the last FC layer CeNN friendly. The accuracy of the network with this downsampling strategy reaches 80.5%. Since this approach does not give as good accuracy as these approaches that change the size of feature map discussed above, we do not include it in the benchmarking effort discussed in Section 6.

5 CENN ARCHITECTURES

In this section, we introduce our CeNN-based architecture for realizing CeNN-friendly CoNNs. Our architecture is general and programmable for any CoNN that contains convolution, ReLU, and pooling layers. Meanwhile, by changing the configurations (e.g., SRAM size, number of OTAs) and parameters of the circuits (e.g., bias current), our CeNN architecture design could be used to satisfy different precision requirements for the network. Thus, we can explore tradeoffs among accuracy, delay, and energy efficiency within the same network. We first present our CoNN-based architecture in Section 5.1. We then describe each component in the architecture, i.e., CeNN cells in Section 5.2, analog memories in Section 5.3, and SRAM in Section 5.4. We also highlight the dataflow for the CoNN network computation using CeNN architecture. In Section 5.5, we discuss the need for ADCs and digital circuitry to support computations in an FC layer (i.e., to support networks as discussed in Section 4.1). Finally, we discuss the programming mechanism for the CeNN templates of the architecture. Throughout we also highlight differences between CeNN cell designs presented here as compared to previous work (e.g., Reference [19]).

5.1 Architecture

Our CeNN architecture for (Figure 5) CoNN computation consists of multiple CeNN arrays (boxes labeled by *CeNN* array *i*). These arrays are the key components for implementing convolution, ReLU and pooling operations in a CoNN. Within each array, there are multiple cells per Section 2.1. The array size can usually accommodate all the image pixels to enable parallel processing of a whole image (extra cells will be power gated to save power). For large images, time multiplexing is used to sequentially process part of the image. The connections between these cells follow the typical CeNN array design as described in Section 2.1. An SRAM array (the rectangle at the bottom of Figure 5) is used to store the templates needed for the CeNN computation. How to configure the CeNN templates with the SRAM data is discussed in Section 5.4. An analog memory array (boxes

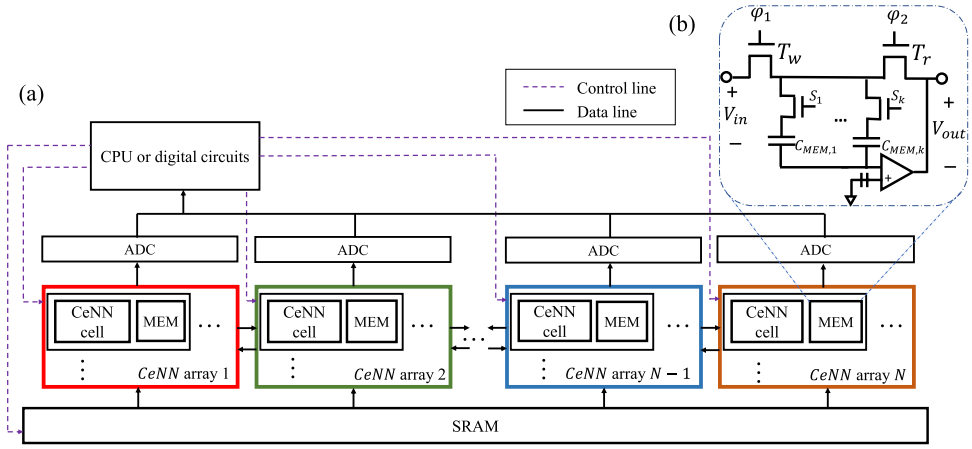


Fig. 5. (a) CeNN-based architecture for CoNN operations; (b) analog memory cell schematic.

labeled by MEM) is embedded into each CeNN cell. The analog memory array is used to store intermediate results for the CeNN computation. Each CeNN array is associated with an ADC. The output of the ADC connects to the host processor or a digital logic, which supports computations for FC layers.

Each CeNN array performs computations associated with one feature map at one time. Thus, N feature maps could perform computations simultaneously with N CeNN arrays. Generally in a state-of-the-art CoNN design, there may be hundreds of feature maps. However, it is not possible to accommodate hundreds of CeNNs in a chip due to area and power restrictions. Therefore, these CeNNs need to be time multiplexed to compute different feature maps in one layer, and the intermediate data needs to be stored in the associated analog memory for processing in the next layer. Thus, the number of CeNN arrays should be chosen to balance the power/area of the chip and the degree of parallel computation of feature maps (FMs) in any given layer.

We use a convolution layer as an example to illustrate how the computation is performed, since it is typically the most time/energy consuming layer in state-of-the-art CoNN designs. We assume layer L_l is a convolution layer, and the layer has C_{l-1} feature maps as inputs and C_l feature maps as outputs. We assume the number of CeNN arrays is N. For each output feature map $FM(l,i)$ in layer L_l, the computation required is shown in Equation (12). Namely, each feature map j (j from 1 to C_{l-1}) in layer L_{l-1} must convolve with a kernel $K(l,j,i)$, and the sum of the convolution results need to be computed. That is,

$$FM(l,i) = \sum_{j=1}^{C_{l-1}} K(l,j,i) * FM(l-1,j). \tag{12}$$

The computation in Equation (12) needs to be repeated C_l times to obtain the results for all the feature maps in Layer l.

To compute feature map $FM(l,i)$, we first perform convolution operations on N feature maps in layer $l-1$ from $FM(l-1,1)$ to $FM(l-1,N)$, to obtain $FM^{(1)}_{temp}$ to $FM^{(N)}_{temp}$ (i.e., $FM^{(N)}_{temp} = K(l,N,i) * FM(l-1,N)$). Then we perform $FM^{(1)}_{pSum} = \sum_{i=1}^{N} FM^{(i)}_{temp}$ by leveraging the connections among these CeNNs. The intermediate results $FM^{(1)}_{pSum}$ are stored in the analog memories associated with the CeNN array 1. Similarly, the convolution operation on another N feature maps in layer $l-1$ ($FM(l-1,N+1)$ to $FM(l-1,2N)$) are performed. Again, we compute $FM^{(2)}_{pSum}$ and

store it in the analog memories associated with CeNN array 2. We repeat the above process until all the input feature maps convolved with a convolution kernel, and their partial sums (from $FM_{pSum}^{(1)}$ to $FM_{pSum}^{(M)}$, where $M = C_l/N$) are all stored in the analog memories associated with $CeNN_1$ to $CeNN_M$. If the number of CeNNs, N, is less than M, then one CeNN would store more than one feature maps. Then, we sum these partial sums up to obtain the feature map i in layer L_l (i.e., $FM(l,i) = \sum_{q=1}^{M} FM_{pSum}^{(q)}$). Again, the above process is repeated C_l times to obtain all feature maps in layer L_l. The detailed algorithm is shown in Algorithm 1. Other types of CoNN layer computations are also summarized in the Algorithm 1. By iteratively using the CeNN architecture, we realize different functionalities. The relation between the processing time and number of CeNNs for a convolutional layer l can be calculated as in Equation (13),

$$t = \sum_{l=1}^{l=L} \left[\left(\frac{C_l C_{l-1}}{N-1} + \frac{C_l C_{l-1}}{N} \right) (t_{CeNN} + t_{prog}) + \frac{C_l C_{l-1}}{2(N-1)} t_{MEM-read} + \frac{C_l C_{l-1}}{2(N-1)} t_{MEM-write} \right]. \tag{13}$$

Here, t_{CeNN} refers to the settling time of an CeNN array, and $t_{MEM-read}$ and $t_{MEM-write}$ are the analog memory read and write time, respectively. t_{prog} refers to the reprogramming time of CeNN (i.e., loading new templates).

In our architecture, the reprogramming or reconfiguration overhead mainly includes reading the bit cells from the SRAM block, and using these outputs to control the switches that power gate OTAs to realize different weight values. The overhead of reading bit cell from the SRAM block dominates. The delay and energy of reading data from the SRAM is accounted for in the evaluation section.

The templates of each CeNN can be programmed to implement different kernels in a given CoNN. Before each CeNN operation, all the OTAs must be reconfigured to implement different templates. These templates are read from the SRAM block, where all template values are stored. The bitline outputs of the SRAM are connected to the switches of the OTAs. After configuration, CeNN operations are performed. Below, we discuss the key blocks in the CeNN architecture.

5.2 CeNN Cells Design

CeNN arrays are the core computational elements in our architecture. The CeNN template values for different layers are determined during the network design phase. For convolutional layers, the templates are the same as weights, which are *trained* by deep neural network frameworks. The templates for ReLU and pooling are discussed in Section 3, and they are independent of the specific problem instance. These template values are read from the SRAM to configure the VCCSs in the CeNN cells. Note that all the cells in an array share the same template values. However, different CeNN arrays may employ the same templates (i.e., for ReLU and pooling layers) or employ different templates (i.e., for convolution layers).

Many prior works have focused on CeNNs implemented by analog circuits using CMOS transistors. Per Section 2, a widely used implementation is based on OTAs [9]. Here, an OTA is built with two-stage operational amplifiers [40]. We use N OTAs with quantized g_m values (i.e., g_{m0}, $2g_{m0}, \ldots, 2^{N-1}g_{m0}$) to realize N-bit templates (i.e., weights). The g_{m0}'s values are set according to the power requirement, since g_m's values are proportional to the bias current. Each OTA is connected to a switch for power gating. By power gating different combinations of these OTAs (as shown in Figure 6), different template values can be realized.

The cell resistance (R_{cell} in Figure 1) here is set as $1/g_m$ ($g_m = 2^N g_{m0}$) such that the cell voltage x settles to the desired output to achieve correct CoNN functionality. The cell capacitance (C_{cell}

ALGORITHM 1: CoNN layer computation with CeNN

procedure CeNNForCoNN(K, $FM(l-1,j), \forall j \in \{0,1,\ldots,C_{l-1}-1\})$, L_l)
▷ K are template values in the layer, $FM(l-1,j)$, $(\forall j \in \{0,1,\ldots,C_{l-1}-1\})$ are feature maps from the last layer, L_l is the type of layer l
  if layer L_l = CONV **then** ▷ perform computations in convolution layers
    for i=0 to $C_l - 1$ **do** ▷ compute each feature map $FM(l,i)$ in layer L_l
      for q=0 to $\frac{C_{l-1}}{N} - 1$ **do** ▷ compute convolution on all feature maps in layer L_{l-1}
        for j=0 to $N-1$ **do**
          ▷ multiplications processed in parallel, summations processed in series
          $FM_{pSum}^{(q)} = \sum_{j=1}^{N} K(l, q*N+j, i) * FM(l-1, q*N+j))$
        end for
      end for
      $FM(l,i) = \sum_{j=1}^{C_{l-1}} FM_{pSum}^{(q)}$
    end for
  end if
  if layer L_l = ReLU **then** ▷ compute ReLU on all feature maps
    for q=0 to $\frac{C_{l-1}}{N} - 1$ **do**
      for j=0 to $N-1$ **do**
        ▷ N FMs are processed in parallel, steps in ReLU are performed in series
        $Intermediate(i+q*N) = K(SHIFTLOW) * FM(l-1, j+q*N)$
        $FM(l, j+q*N) = K(SHIFTBACK) * Intermediate(i+q*N)$
      end for
    end for
  end if
  if layer L_l = Pooling **then** ▷ compute pooling on all feature maps
    for q=0 to $\frac{C_{l-1}}{N} - 1$ **do**
      for j=0 to $N-1$ **do**
        for p=0 to 3 **do** ▷ for each neighbor of the current pixel (see Section 3.3)
          $DIFF(p) = K(DIFF(p)) * FM(l-1, j)$
          $Increase(p) = K(INC) * DIFF(p)$
          $Mult(p) = K(MULT) * Increase(p)$
          $FM(l,j) = FM(l-1,j) + Mult(p)$
        end for
      end for
    end for
  end if
end procedure

in Figure 1) is the summation of the output capacitance of nearby OTAs. The delay and energy estimation of a CeNN cell in this article is different from that in Reference [19] in that (1) 32nm technology is used for the hardware design, (2) the $g'_m s$ of the OTAs are larger for faster processing while still satisfying a given power requirement, and (3) the cell resistance R_{cell} in Reference [19] is assumed to be the absolute value of the sum of $g'_m s$, which leads to much larger settling times. Therefore, the work in Reference [19] is a conservative estimation and overestimates the delay and the energy.

5.3 Analog Memory Design

To support operations that may require multiple (analog) steps associated with different CeNN templates, each CeNN cell is augmented by an embedded analog memory array [7] (see Figure 5).

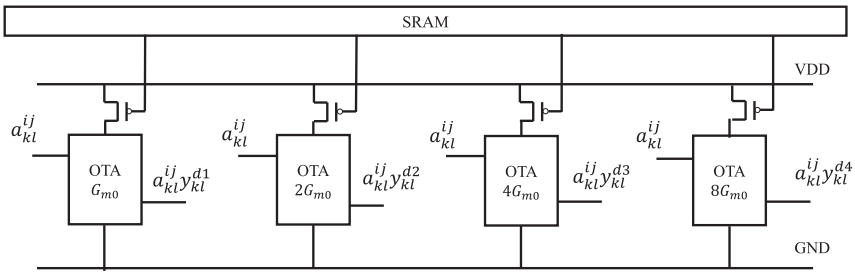


Fig. 6. The schematic for an OTA within each CeNN cell for representing 4-bit weights. Data from SRAM connect to the transistor switches to power gate OTAs so as to program the CeNN template to different values.

For the CeNN based convolution computation described in Section 5.1, analog memory is used to store the intermediate result after each step. For a convolution layer, all the intermediate results described in Algorithm 1 need to be stored in the analog memory. The design of the analog memory and the op amp are from Reference [7]. Specifically, the analog memory array is implemented by a write transistor (T_w) and read transistor (T_r) to enable write and read. An additional op amp is used to hold the state of the capacitors shown in Figure 5(b). Multiple pass transistors and capacitors C_{mem} are used to store data. Each capacitor C_{mem} and pass transistor forms a memory cell (as a charge storage capacitor) within the analog memory array that could store one state value of a CeNN cell (i.e., data correspond to one pixel). The number of capacitors (C_{mem}) within one analog memory array depends on the data needed to store in the memory. The gates of the pass transistors are connected to a MUX. Thus, V_{s1} to V_{sx} shown in Figure 5(b) are controlled by the MUX to determine which capacitor memory needs to be written/read. If the analog signal needs to write to the memory, then transistor T_w is on, and one of the pass transistors is selected by the MUX. The data are written to the corresponding capacitor C_{mem}. For a read, transistor T_r is on, and one of the pass transistors is selected by MUX. As each analog memory array is dedicated to one CeNN cell, CeNN cells can access these memory arrays in parallel.

5.4 SRAM

An SRAM array is used to store all the template values required for CeNNs to realize a CoNN. While the SRAM itself is a standard design, we still need to carefully select the number of bitlines within one word line due to power and performance constraints. One design choice may have one word containing all the template values for one CeNN array. For one template operation, $10N_b$ bits are needed for N_b-bit precision weight (including nine template values and a bias). For this option, if N CeNN arrays have distinct sets of templates (i.e., in the convolution layer), then N accesses will be required. However, if N CeNN arrays have the same templates (i.e., in the ReLU and pooling layers), then only one access is required. To reduce the number of accesses, two or more $10N_b$-bit words may be accessed in one cycle by using either more read ports or longer SRAM words. After SRAM cell data are read, they are used to control how an OTA is power gated, which in turn realizes different weight values.

5.5 ADC and Hardware for FC Layers

Each CeNN is connected to an ADC to convert analog data to a digital representation whenever necessary, e.g., for FC layer computations (i.e., the last layer in Figure 2 computation). FC layers typically require computing the dot product of two large vectors. Such operations are not well-suited for CeNNs with limited kernel size. Hence, a CPU, GPU, or other hardware should

be employed. In the benchmarking efforts to be discussed in Section 6, combinations of digital adders, multipliers, and registers (i.e., ASICs) are used. For simplicity, ripple carry adders and array multipliers are employed in our simulations. Both inputs and weights are N_b bits (where N_b refers the to the precision of CeNN). We also assume that the weights for the FC layer are stored in SRAM. The result of the multiplication is $2N_b$ bits, while an additional N_b bits are used to store the final results of this layer to avoid overflow. Thus, there are $3N_b$ bits at the output. That said, alternative network designs as shown in Section 4.2 can eliminate this layer.

6 EVALUATION

We now evaluate the architectures, networks, and algorithms described above to determine (i) whether or not CeNN-friendly CoNNs are competitive with respect to existing architectures and algorithms that address the same dataset and, (ii) if so, what elements of the CeNN design space lead to superior application-level FOM (e.g., energy and delay per classification and accuracy). While our CeNN architecture can be applied to different datasets, we specifically compare our approach to other efforts in the context of the MNIST and CIFAR-10 dataset given the wealth of comparison points available.

6.1 Simulation Setup

Components of the CeNN-based architecture are evaluated via SPICE simulation using the Arizona State University Predictive Technology Model (ASU PTM) for high-performance MOSFET devices at the 32nm technology node [65]. We use CACTI 7 [3] to estimate the delay and energy needed for SRAM accesses with the same technology node. The size of SRAM is set as 16KB to retain reasonable access time/energy, while also accommodating all templates for the proposed networks. The SRAM can be scaled if necessary to accommodate all the weights in larger networks. In our SRAM design, each wordline contains $10N$ bitlines, so that all weights needed for one CeNN operation can be read from SRAM only once. The analog memory is also scaled to the same technology node.

Though the architecture itself can realize any number of bits, we assume 4-bit and 8-bit precision in our evaluation. Four-bit results help to inform the energy efficiency of our design with reasonable application-level classification accuracy, while 8-bit designs generally do not sacrifice accuracy when compared with 32-bit floating point representation. We use four CeNNs that correspond to four feature maps in the networks described in Section 4 for evaluation. However, the number of CeNNs could be changed as a tradeoff between processing time and area/power, as discussed in Algorithm 1 in Section 5.1. We take the trained model from TensorFlow, and perform inference computations in a MATLAB based infrastructure with both feature maps and weights quantized to 4 bits or 8 bits to predict accuracy.

The supply voltage is set to 1 V, and the ratio of the current mirrors in the OTAs is set to 2 to save power in the first stage of OTA. For different precision requirements, the same OTA schematic is used with different transistor sizes and bias currents. The multiple OTA design in Section 5.2 could be used to represent different number of bits for weights. These OTAs are reprogrammed in each step. Here, for each OTA, we convert the signal-to-noise ratio (SNR) of OTA to bit precision using the methods in Reference [27] to represent different number of bits for feature maps. Compared to the 4-bit designs, the 8-bit designs increase the bias current by 7.5× and increase the transistor width by 4× to increase the SNR of the circuit from 32.1dB to 50.6dB. Thus, the delay increases by 4.3× due to the change of bias conditions and increase of transistor size (i.e., parasitic capacitance increases), and the power increases by 7.5× as the bias current increases. The g_m's of an OTA can be selected to tradeoff processing speed and power. Here, we use four OTAs with g_m values 12μA/V, 24μA/V, 48μA/V, and 96μA/V to realize 4-bit templates (i.e., weights). In the 8-bit design, larger granularity is used, and g_m values are set to 0.75μA/V, 1.5μA/V, 3μA/V, 6μA/V, 12μA/V, 24μA/V,

48μA/V, and 96μA/V. We assume state-of-the-art ADC designs [60, 62] to estimate the delay and energy of analog to digital conversion needed before the FC layer in the network in Figure 2. We assume each CeNN is associated with an ADC to convert analog data to digital representation.

We employ the same device model to benchmark analog memory arrays. We first determine the capacitance and size of pass transistors based on the methods in Reference [7]. The capacitance is $C_{mem} = 55fF$ and the width of the transistor is 180nm. We use a minimum length of 30nm. Then, memory write time is determined by the resistance of pass transistor T_{pass} and the capacitor C_{mem}. The memory read time is determined by the analog signal through the read buffer. We use SPICE to measure the delay of the analog memories. Per simulations, each memory write and read requires 124ps and 253ps, respectively.

To satisfy the precision requirements, we also study the robustness of our architecture by evaluating the PVT condition with four corner cases (FF 80°C, SS −40°C, FS 27°C, SF 27°C). We also apply a 5% variation on the supply voltage to study the impact to the OTA in the CeNN cell, which is the essential computational element in our design. Since the g_m of the OTAs in CeNN cell represents the template values in the CeNN operations, we evaluated the g_m variations in the OTA design in these corner cases in the PVT condition study. We specifically focus on the OTA with the largest g_m value in our design, since the variation of that OTA will have the largest impact on the multiplication results. Our simulation results show that in the worst corner case, the error of the circuit still satisfies the precision requirements. Regarding parasitic capacitance, we have not yet completed a layout of the architecture and cannot precisely model the impact of parasitic capacitance. However, (1) parasitic capacitances within a cell are smaller than the cell capacitance C_{cell} shown in Figure 5(b), and (2) we assume a CeNN has only local connections with radius of 1 (i.e., to implement 3×3 kernels). Thus, we expect the interconnect parasitic capacitance to be small as a given cell is only connected to its immediate neighbors.

6.2 Evaluation of the CeNN Based Architecture

We initially use the 4-bit CeNN design as an example to show how we evaluate the accuracy, delay, and energy of our CeNN architecture for performing CoNN computations. We use MNIST as the benchmarking dataset, and the network in Figure 2 and the network in Figure 3 with different configurations (summarized in Table 1) are used for evaluation. Eight-bit results are also presented here.

We first measure the energy and delay associated with each layer of a CeNN-friendly CoNN for the 4-bit design. Table 3 summaries the delay and energy for each layer for the networks in Figure 2 and Figure 3. Per Table 3, the energy for each layer in the network in Figure 3 decreases with subsequent layers as data are down-sampled, and only a subset of cells in a CeNN are used for the computation. However, delay remains constant (for each layer) as all computations in CeNN cells occur in parallel. (The network in Figure 3 has a higher latency than the network in Figure 2 in the CeNN components due to the fact that more layers are employed to properly downscale the image, i.e., more template operations are required.) We use the MATLAB framework to quantize the weights and inputs to 4 bits in the inference stage and classification accuracies for each design are shown in Table 4. We find that for all cases, the accuracy decreases about 2% for each design compared with the 32-bit floating point design shown in Table 1, due to the reduced precision of input and weights for our simple network.

We next consider the impact of the ADCs and the FC layer. The delay and energy for an ADC can be approximated based on a 28nm SAR ADCs design from Reference [60]. The total time and energy to port all analog data to the digital domain for the network in Figure 2 are 166.7ns and 3,834pJ, respectively (using time multiplexing). For the FC layer, we first use the uniform beyond-CMOS benchmarking (BCB) methodology [45] to estimate the delay and energy for a full adder

Table 3. Delay and Energy for Each CeNN Layer

	Network in Figure 2		Network in Figure 3	
Layer	Delay (ns)	Energy (pJ)	Delay (ns)	Energy (pJ)
Conv. 1	5.3	626	5.3	626
ReLU1	10.7	536	10.7	536
Pooling1	85.5	4,290	85.5	3,398
Conv. 2	42.8	2,827	42.8	981
ReLU2	10.7	410	10.7	186
Pooling2	85.5	3277	85.5	1489
Conv. 3	—	—	42.8	519
ReLU3	—	—	10.7	115
Pooling3	—	—	85.5	921
Conv. 4	—	—	53.4	582
ADC + FC	291.1	7,875	—	—
Total	531.6	19,841	432.9	9,353

Table 4. Accuracy, Delay and Energy with 4-bit CeNN Architecture Design

	Network in Figure 2			Network in Figure 3		
Approach	Accuracy	Delay	Energy	Accuracy	Delay	Energy
Baseline	96.5%	532ns	19.8nJ	96.0%	433ns	9.4nJ
Average Pooling	95.7%	372ns (1.4×)	12.5nJ (1.5×)	94.3%	192ns (2.2×)	4.4nJ (2.1×)
Nonlinear operation	92.9%	357ns (1.5×)	12.0nJ (1.7×)	91.5%	116ns (3.7×)	3.4nJ (2.8×)

as well as the register for storing temporary data during the computation. Then, we estimate the delay of multiplication and addition operations by counting the number of full adders in the critical path of the multiplier and adder. The energy per operation is estimated by the summation of all full adder operations and loading/storing data during computation. The energy and delay overhead due to the interconnect parasitics is also taken into account by using the BCB methodology. Overall, the delay and energy of the FC layer are 124.4ns and 4,041pJ, and they contribute 23% and 20% to the total delay and energy per classification for the network in Figure 2 (including ADCs), respectively.

Though the network in Figure 3 (with no FC layer) requires additional layers to properly downscale the image, the delay is still 19% lower than the network in Figure 2. Additionally, the network in Figure 3 requires 2.1× less energy per classification due to downsampling. However, the accuracy for the network in Figure 3 is 0.5% lower than that in Figure 2.

To evaluate the impact of different approaches for pooling operations, as well as how non-linear template operations impact energy, delay, and accuracy, we apply each design alternative to the networks in Figures 2 and 3. Results are summarized in Table 4. The numbers in parenthesis refer to the comparison between the alternative approach with the baseline (i.e., the network in Figures 2 and 3 with maximum pooling and linear templates). By using average pooling, the delay/energy is reduced by 1.4×/1.5× and 2.2×/2.1× for the networks in Figures 2 and 3, respectively—as 16 CeNN steps are reduced to 1 step. The accuracy is reduced by 0.8% for the network in Figure 2 and 1.7% for the network in Figure 3, respectively. Designs with non-linear templates lead to reductions in delay/energy of 1.5×/1.7× and 3.7×/2.8× for the networks in Figures 2 and 3, respectively—as both

Table 5. Accuracy, Delay, and Energy with 8-bit CeNN Architecture Design

	Network in Figure 2			Network in Figure 3		
Approach	Accuracy	Delay	Energy	Accuracy	Delay	Energy
Baseline	98.0%	1442ns	104.9nJ	97.8%	1828ns	56.6nJ
Average pooling	97.5%	773ns	49.9nJ	97.4%	819ns	23.0nJ
Nonlinear operation	95.4%	710ns	46.2nJ	94.2%	490ns	23.6nJ

ReLU and pooling operations are reduced to a single step. However, the accuracy drops by 3.6% and 4.5%, respectively, following the same trend as the floating point precision.

It is obvious that the accuracy drops for 4-bit designs (in Table 4) compared with 32-bit floating point designs (in Table 1). Meanwhile, there is evidence that the 8-bit precision for many networks usually do not sacrifice accuracy compared with 32-bit floating point design and are widely used in the state-of-the-art training and inference engine [24]. Therefore, we also evaluate accuracy, delay, and energy for our 8-bit CeNN design using the same method above to show the tradeoffs. In this design, we use OTAs with an SNR equivalent to 8-bit precision. The weights are also set to 8 bits. We use a different design [26] to evaluate ADC overhead to reflect converting analog signals to 8-bit digital signals. The inputs and weights of the digital FC layer are also set to 8 bits. The results are summarized in Table 5. As expected, the delay and energy both increase compared to the 4-bit design by 2.0–4.2× and 3.8–7.5× depending on the specific designs, but the accuracy approaches that of 32-bit floating point data. In this design, the delay and energy of network in Figure 3 increase more than that of the network in Figure 2. The computations of the network in Figure 3 is mostly in the analog domain, while the computations in the network in Figure 2 use both analog and digital circuits. As the number of bit increases, the delay and energy for computations associated with analog circuits increase generally faster than the delay and energy for computations associated with digital circuits.

6.3 Comparison to Other MNIST Implementations

It now begs the question as to how our CeNN-based approach compares to other accelerator architectures and algorithms that have been developed to address classification problems such as MNIST. Since the computations in our designs are mostly performed in analog domain, we first compare our work with a recent logic-in-memory analog implementation that addresses the same problem [5]. We compare the delay and energy of convolution layers here. As Reference [5] only reports the throughput and energy efficiency for the first two convolutional layers in LeNet-5, using 7-bit inputs and 1-bit weights, we also use the throughput and energy efficiency for convolution layers in our baseline network design for fair comparison. The comparison results are shown in Table 6. Our CeNN design demonstrates 10.3× EDP improvements over those in Reference [5]. At the application level, we still obtain better classification accuracy (96.5% v.s. 96%). However, since Reference [5] does not include the data for FC layer, they do not have the complete EDP data on MNIST. Hence, we do not include the implementation in Reference [5], the benchmarking plot (Figure 7), to be discussed.

We next consider a state-of-the-art digital DNN engine presented in Reference [59] with 28nm technology node for the MNIST dataset at iso-accuracy with our CeNN based design. We scale the design in Reference [59] from 28nm to 32nm for a fair comparison using the method described in Reference [49]. The work in Reference [59] assumes an multilayer perception (MLP) network with 8-bit feature maps and weights, varying the different network sizes. Among these different networks, we find three implementations that match the accuracy of our three designs. Their network sizes are 784×16 × 16 × 16×10, 784×32 × 32 × 32×10, and 784×64 × 64 × 64×10, with accuracy of

Table 6. Detailed Comparison to Analog Implementation [5] for MNIST Dataset

Approach	Precision of feature maps	Precision of weights	Efficiency	Energy Efficiency	Technology	Accuracy
CeNN-based approach	4 bits	4 bits	251 GOPS	12.3 TOPS/W	32nm	96.5%
Logic-in-memory analog circuit [5]	7 bits	1 bit	10.7 GOPS	28.1 TOPS/W	65nm	96%

Table 7. Detailed Comparison to DNN Engine [59] for MNIST Dataset

Comparison	Approach	Accuracy	Bits	Delay (ns)	Energy (nJ)	EDP (nJ-ns)
Comparison 1	CeNN—Network in Figure 3, baseline	96.03%	4	372	9.0	4.6×10^3
	DNN engine [59]	95.41%	8	1,001	39.9	4.0×10^4
Comparison 2	CeNN—Network in Figure 2, baseline	96.5%	4	532	19.8	1.1×10^4
	DNN engine [59]	97.0%	8	1,478	72.5	1.0×10^5
Comparison 3	CeNN—Network in Figure 3, avg. pooling	95.41%	8	810	230	1.9×10^5
	DNN engine [59]	97.58%	8	2,692	145	3.9×10^5

95.41%, 97.0%, and 97.58%, respectively. Meanwhile, our three designs are (i) network in Figure 3, baseline with 4-bit precision (accuracy to be 96.03%); (ii) network in Figure 2, baseline with 4-bit precision (96.5% accuracy); and (iii) network in Figure 3, average pooling with 8-bit precision (97.41% accuracy). We compare FOMs including energy and delay at iso-accuracy for these designs.

From Table 7, we can find that in our implementation, the EDP and energy efficiency are 2.1–8.7× and 6–27× better, respectively, than the DNN engine [59]. The 8-bit CeNN based design is not as efficient as the 4-bit design with respect to energy efficiency—compared with the DNN engine due to the fact that analog circuits have worse area/delay/energy compared with digital circuits in higher precision. Here, our delay and energy data are based on simulations, while the data for DNN engine is based on the measurement. Therefore, some discrepancy may exist. However, in general, with the CeNN approach, (i) high parallelism can be achieved in terms of multiplications and additions in the CeNN-based architecture, (ii) the network exploits local analog memory for fast processing, and (iii) accessing feature maps in the analog domain is faster than accessing the digital weights in the digital domain. Thus, the weight stationary approach is used. That said, once the weights are read from the SRAM (i.e., all the cells are configured), all the computations associated with the weights are performed. The weights do not need to be read from SRAM again. Therefore, the total weight access time is minimized. Since there are still unused OTAs in our design, it may be further optimized to reduce the delay and energy.

We also compare our work with a wider range of implementations, including custom ASIC chips [8, 41, 50, 59], neural processing units [18], spiking neural networks [14, 28, 42], crossbar implementations [57], and CPU/GPU-based solutions of the DropConnect approach [58] (the most accurate approach for MNIST to date; data are measured via i7-5820K, 32GB DDR3 with Nvidia Titan). Figure 7 plots the EDP vs. misclassification rate for all these approaches. To make a fair comparison, we again scale all delay/energy data to the 32nm technology node using the ITRS data based on Reference [49].

Note that the comparison is shown in the log scale, additional uncertainties (interconnects parasitics, clocking, control circuits) should not change the overall trend shown in Figure 7 as the EDP of these elements would not be orders of magnitude larger [45]. Our approach has significantly lower EDP compared with other approaches with comparable classification accuracy. Among our designs, higher EDPs are generally correlated with higher accuracy. We draw a Pareto frontier line (the green line in Figure 7 according to the product of misclassification rate and the EDP. In our

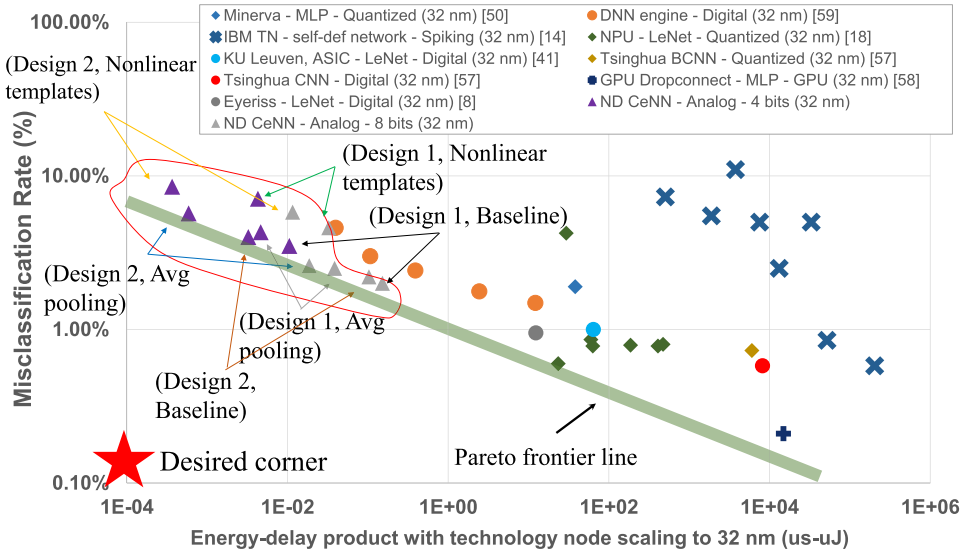


Fig. 7. Benchmarking results for CeNN-friendly CoNNs as well as other algorithms and architectures for the MNIST digit classification problem.

Table 8. Accuracy, Delay, and Energy for Different Noise Levels

Approach	CeNN-friendly AlexNet C96-C256-C384-C384-C256	CeNN-friendly AlexNet C64-C128-C256-C256-C128	CeNN-friendly AlexNet C64-C128-C128-C128-C64
Accuracy	83.9%	82.2%	80.8%
Delay (μs)	311	106	47
Energy (μJ)	497	169	75

designs, several datapoints are on the Pareto frontier. Specifically, for the 4-bit design, the network in Figure 3 with maximum pooling and linear templates, and the network in Figure 3 with average pooling and linear templates are on the Pareto frontier, while for the 8-bit design, the network in Figure 2 with average pooling linear templates are on the Pareto frontier in the plot. We should add that the EDP values of some of the implementations [8, 41, 50, 59] in Figure 7 are obtained from actual measurements, while others are from simulation. Therefore, some discrepancy may exist.

6.4 Evaluation of Larger Networks

In Section 6.3, we discussed a comprehensive comparison using the MNIST problem as the context. However, networks for MNIST are relatively simple. In this subsection, we also compare our CeNN design with other implementations that target larger networks, i.e., we compare with other accelerators that solve the CIFAR-10 problem.

For the CIFAR-10 dataset, images with size 32×32 are used. We also use CeNNs with the same size to enable parallel processing. The evaluation setup is the same as in Section 6.1. We use the networks discussed in Section 4.3 and summarize our results in Table 8. Here, we use 4-bit design to maximize the energy efficiency, and the accuracy is close to 32 floating point accuracy (given in Table 2).

We compare our approach with a large number of implementations available that solve the CIFAR-10 problem. The benchmarking plot is shown in Figure 8. The implementation includes IBM

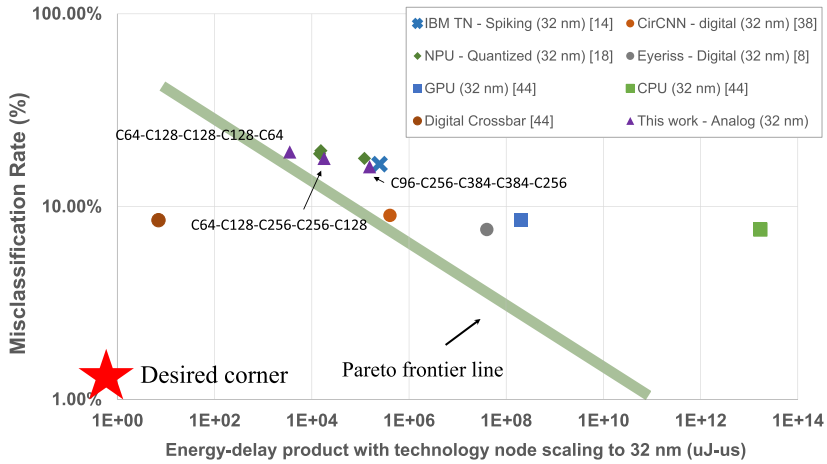


Fig. 8. Benchmarking results for CeNN-friendly CoNNs as well as other algorithms and architectures for the CIFAR-10 classification problem.

Table 9. Detailed Comparison to NPU [18] for the CIFAR-10 Dataset

Approach	Technology node	Accuracy	Bits	Delay (μs)	Energy (μJ)	EDP (μJ-μs)
CeNN-based approach	32nm	80.8%	4	47	75	3,525
NPU [18]	32nm	80.5%	8	485	32	15,332

TrueNorth [14], Fourier transform approach [38], NPU [18], Eyeriss [8], a mixed-signal approach [4], and the CPU and GPU data reported in Reference [44]. We also draw a Pareto frontier line based on the product of misclassification rate and EDP of the data points collected in Figure 8. From the plot, one of our CeNN datapoint (C64-C128-C128-C128-C64) lands on the Pareto frontier.

We also make an iso-accuracy comparison with the NPU data point shown in the plot. We selected a datapoint from our design with similar accuracy to the design in NPU. The detailed comparison is shown in Table 9. Not only is the accuracy of our CeNN design 0.3% better than the NPU approach, but also our design achieves 4.3× EDP compared with the NPU approach. Note that the NPU data are also simulation results.

To articulate our evaluation, we also discuss the differences between our work and other analog accelerators, i.e., ISAAC [53] and RedEye [39] here. Our work differs from ISAAC and RedEye in the following aspects.

(1) Different computation elements are used. ISAAC uses a crossbar architecture, where multiplication and summation are carried out via analog voltage, conductance, and current, and signals are accumulated horizontally in the crossbar rows within the chip. RedEye uses tunable capacitors as computation units. Our approach uses CeNN cells as the base element, where multiplications and partial sum calculations are performed using OTAs within each CeNN cell.
(2) Different dataflows are used. ISAAC uses an in-memory computation architecture, where memristors are used for both storing the weights and performing computation. In RedEye, column-based computation elements are used, and data are passed vertically. In our CeNN architecture, the memory and the computation units are separated. OTAs are used for multiplication while analog memories are used to store intermediate results.

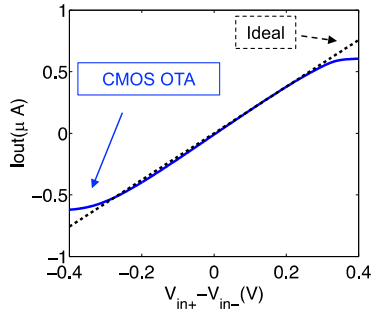


Fig. 9. Ideal I-V characteristics and actual characteristics for OTA design.

Table 10. Classification Accuracy for MNIST When Actual I-V Characteristics Are Included in Training/Inference

Network type	Original network	Inference with actual I-V	Training & inference with actual I-V
Network in Figure 2, linear templates, max pooling	98.1%	96.5%	97.9%
Network in Figure 3, linear templates, max pooling	97.8%	95.8%	97.6%

(3) The requirements on devices are different. In ISAAC, memristors are required to perform the logic-in-memory computation. In RedEye, conventional CMOS is used for benchmarking. While we also assume conventional CMOS for benchmarking activities in this article, our work is compatible with other emerging devices as well (e.g., many emerging devices have been considered in the context of CeNN implementations per Reference [48]).

6.5 Training with Actual I-V Characteristics

In Section 6.2, we show that by leveraging the 8-bit representation, the accuracy does not decrease much compared with the 32-bit floating point representation. However, another source of error comes from the actual I-V characteristics of an OTA. For example, in Figure 9, when the difference of two inputs, $(V_{in+} - V_{in-})$, of the OTA is larger than 0.2V, the mismatch between the actual and ideal I-V characteristics becomes more severe. This behavior could potentially decrease the accuracy.

To study the impact, we include the mismatch described above into the inference stage. We use the actual I-V characteristics of an OTA obtained from SPICE simulation, and build a look-up table. We then embed the table into the MATLAB based CeNN simulator in the inference stage. That is, whenever an OTA operation is required, results for the OTA are read from the lookup table, instead of by direct matrix multiplication. Simulations of the networks in Figures 2 and 3 suggest that by including the actual I-V characteristics in the network, the accuracy decreases from 98.1% and 96.5% to 96.8% and 95.8%, respectively.

However, this accuracy decrease can be largely compensated by leveraging the I-V characteristics in the training stage. We use the same look-up table, and plug it into the forward path of the training stage of the network in the TensorFlow framework. By considering the I-V characteristics during training, the accuracy increases and become close to the ideal accuracy. The results are summarized in Table 10. We can see that by using the actual I-V characteristics in the training stage, the accuracy only decreases 0.2% when compared with the original network for the baseline design for network in Figure 2 and network in Figure 3. This approach should be applicable for other non-ideal circuit behaviors.

Whether individualized training might be needed is still an open question. However, the existing literature suggests that some PVT variations and noise in the circuit may not greatly impact application level accuracy for both MOSFETs and emerging devices (e.g., see References [39, 64]); thus individualized training would not be needed. Researchers have also investigated on-chip training given device variations (e.g., see References [46, 63]), and reasonable application level accuracy results are indeed obtained. Essentially, at present, there are no firm conclusions about whether individualized training will be required. We will also study this in our future work.

7 CONCLUSIONS AND DISCUSSIONS

This article presents a mixed-signal architecture for hardware implementation of convolutional neural networks. The architecture is based on an analog CeNN realization. We demonstrate the use of CeNN to realize different layers of CoNN, and the design of CeNN-friendly CoNNs. We present tradeoffs for each CeNN-based design and compare our approaches with various other existing accelerators to illustrate the benefits for the MNIST and CIFAR-10 problem as case studies. Our results show that the CeNN-based approach can lead to superior performance while retaining reasonable accuracy. Specifically, 8.7× EDP for the MNIST problem and 4.3× EDP for the CIFAR-10 problem are obtained in iso-accuracy comparison, when comparing with state-of-the-art approaches.

Our architecture targets were originally/primarily for edge devices. Network sizes for edge devices (e.g., MobileNet [20], SqueezeNet [21], etc.) are usually much smaller than AlexNet. Thus, AlexNet for CIFAR-10 dataset should be sufficient to illustrate how our approach can be applied to larger networks and how our approach compares other existing works. Furthermore, these networks also only have kernel sizes 3×3 or 1×1, which are suitable for our CeNN computations. We expect that the network model deployed in edge devices should be smaller than our CeNN friendly AlexNet. Thus, our CeNN architecture should be able to process all tasks that could be reasonably processed by IoT devices efficiently. As future work, we will study other larger network topologies to further ensure that reasonable classification accuracies could be obtained (i.e., when compared to published work) and will also consider the CeNN approach with respect to metrics such as energy and delay in the context of these networks.

We will also continue evaluating what benefits machine-learning/computer vision applications can get from analog computation with both MOSFETs and emerging devices.

REFERENCES

[1] [n.d.]. Official site of the Toshiba SPS 02 Smart Photosensor. Retrieved from http://www.toshiba-teli.co.jp/en/products/industrial/sps/sps.htm.

[2] [n.d.]. Software Library for Cellular Wave Computing Engines in an era of kilo-processor chips Version 3.1. Retrieved November 29, 2016 from http://cnn-technology.itk.ppke.hu/Template_library_v3.1.pdf.

[3] Rajeev Balasubramonian, et al. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *Trans. Arch. Code Optim.* 14, 2 (2017).

[4] Daniel Bankman, et al. 2018. An always-on 3.8 μJ 86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS. In *Proceedings of the International Solid-State Circuits Conference (ISSCC'18).* 222–224.

[5] Avishek Biswas, et al. 2018. Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-Based machine learning applications. In *Proceedings of the International Solid-State Circuits Conference (ISSCC'18).* 31.1.

[6] Y-Lan Boureau, et al. 2010. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the International Conference on Machine Learning (ICML'10).* 111–118.

[7] R. Carmona-Galan, et al. 1999. An 0.5 μm CMOS analog random access memory chip for TeraOPS speed multimedia video processing. *IEEE Trans. Multimedia* 1, 2 (1999), 121–135.

[8] Yu-Hsin Chen, et al. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *J. Solid State Chem.* 52, 1 (2017), 127–138.

[9] Eric Y. Chou, et al. 1997. VLSI design of optimization and image processing cellular neural networks. *IEEE Trans. Circ. Syst. I* 44, 1 (1997), 12–20.

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

[11] Leon O. Chua and Lin Yang. 1988. Cellular neural networks: Applications. *IEEE Trans. Circ. Syst.* 35, 10 (1988), 1273–1290.

[12] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. 2013. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 8609–8613.

[13] Jia Deng, et al. 2009. Imagenet: A large-scale hierarchical image database. *Comput. Vis. Pattern Recogn.* (2009), 248–255.

[14] Steve Esser, et al. 2015. Backpropagation for energy-efficient neuromorphic computing. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'15)*. 1117–1125.

[15] Christian Szegedy, et al. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[16] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'13)*. 6645–6649.

[17] Song Han, et al. 2016. EIE: Efficient inference engine on compressed deep neural network. *Proceedings of the International Symposium on Computer Architecture (ISCA'16)*.

[18] Soheil Hashemi, et al. 2017. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *Proceedings of the Annual Conference on Design, Automation, and Test in Europe (DATE'17)*. 1474–9.

[19] Andras Horvath, et al. 2017. Cellular neural network friendly convolutional neural networks CNNs with CNNs. In *Proceedings of the Annual Conference on Design, Automation, and Test in Europe (DATE'17)*. 145–150.

[20] Andrew G. Howard, et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint* 1704.04861 (2017).

[21] Forrest Iandola, et al. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size. *arXiv preprint* (2016).

[22] Jesus E. Molinar-Solis, et al. 2007. Programmable CMOS CNN cell based on floating-gate inverter unit. *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*. 49, 1 (2007), 207–2016.

[23] Nicola Jones. 2017. Machine learning tapped to improve climate forecasts. *Nature* 548, 7668 (2017), 379–380.

[24] Norman P. Jouppi, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture (ISCA'17)*.

[25] Min-Joo Kang and Je-Won Kang. 2016. Intrusion detection system using deep neural network for in-vehicle network security. *PloS One* 11, 6 (2016), e0155781.

[26] John P. Keane, et al. 2017. 16.5 An 8GS/s time-interleaved SAR ADC with unresolved decision detection achieving 58dBFS noise and 4GHz bandwidth in 28nm CMOS. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC'17)* (2017), 284–285.

[27] Walt Kester. 2009. Understand SINAD, ENOB, SNR, THD, THD+ N, and SFDR so you don't get lost in the noise floor. *MT-003 Tutorial* (2009).

[28] J. K. Kim, et al. 2015. A 640M pixel/s 3.65mW sparse event-driven neuromorphic object recognition processor with on-chip learning. In *VLSI Circuits*. 50–51.

[29] Kwanho Kim, Seungjin Lee, Joo-Young Kim, Minsu Kim, and Hoi-Jun Yoo. 2009. A 125 GOPS 583 mW network-on-chip based parallel processor with bio-inspired visual attention engine. *IEEE J. Solid-State Circ.* 44, 1 (2009), 136–147.

[30] D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv: 1412.6980* (2014).

[31] Matej Kristan, et al. 2017. The visual object tracking vot2013 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 1949–1972.

[32] Alex Krizhevsky, et al. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'12)*. 1097–1105.

[33] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*).

[34] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[35] Yann Lecun, Leon Bottou, Yoshua Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov. 1998), 2278–2324.

[36] Yann LeCun, Corinna Cortes, and C. J. Burges. 2010. MNIST handwritten digit database. AT&T Labs [Online]. Retrieved from http://yann.lecun.com/exdb/mnist.

[37] Lei Wang, et al. 1998. Time multiplexed color image processing based on a CNN with cell-state outputs. *IEEE Trans. VLSI* 6, 2 (1998), 314–322.

[38] Siyu Liao, et al. 2017. Energy-efficient, high-performance, highly-compressed deep neural network design using block-circulant matrices. In *Proceedings of the 36th International Conference on Computer-Aided Design*. 458–465.
[39] Robert LiKamWa, et al. 2016. RedEye: Analog ConvNet image sensor architecture for continuous mobile vision. *ACM SIGARCH Comput. Arch.* 6, 1 (2016).
[40] Qiuwen Lou, et al. 2015. TFET-based operational transconductance amplifier design for CNN systems. In *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI'15)*. 277–282.
[41] Bert Moons, et al. 2016. A 0.3-2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets. In *VLSI Circuits*. 1–2.
[42] Hesham Mostafa, et al. 2017. Fast classification using sparsely active spiking networks. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS'17)*. 1–4.
[43] Ihab Nahlus, et al. 2014. Energy-efficient dot product computation using a switched analog circuit architecture. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'14)*. 315–318.
[44] Leibin Ni, et al. 2017. An energy-efficient digital ReRAM-crossbar-based CNN with bitwise parallelism using block-circulant matrices. *IEEE J. Explor. Solid-State Comput. Devices Circ.* 3 (2017), 37–46.
[45] D. E. Nikonov, et al. 2015. Benchmarking of beyond-CMOS exploratory devices for logic integrated circuits. *IEEE J. Explor. Solid-State Comput. Dev. Circ.* 1 (2015), 3–11.
[46] Xiaochen Peng Pai-Yu Chen and Shimeng Yu. 2017. NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures. In *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 6–1.
[47] Indranil Palit, et al. 2015. Analytically modeling power and performance of a CNN system. In *Proceedings of the IEEE-International Conference on Control, Automation and Diagnosis (ICCAD'15)*. 186–193.
[48] Chenyun Pan and Azad Naeemi. 2016. Non-Boolean computing benchmarking for beyond-CMOS devices based on cellular neural network. *IEEE J. Explor. Solid-State Comput. Dev. Circ.* 2 (2016), 36–43.
[49] Robert Perricone, et al. 2016. Can beyond-CMOS devices illuminate dark silicon? In *Proceedings of the Annual Conference on Design, Automation, and Test in Europe (DATE'16)*. 13–18.
[50] Brandon Reagen, et al. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the International Symposium on Computer Architecture (ISCA'16)*. 267–278.
[51] Angel Rodríguez-Vázquez, et al. 2004. ACE16k: The third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs. *IEEE TCAS I: Regul. Pap.* 51, 5 (2004), 851–863.
[52] Tamas Roska and Leon O. Chua. 1993. The CNN universal machine: An analogic array computer. *IEEE Trans. Circ. Syst. II* 40, 3 (1993), 163–173.
[53] Ali. Shafiee, et al. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ACM SIGARCH Computer Architecture* (2016), 14–26.
[54] David Silver, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354.
[55] Karen Simonyan, et al. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint* (2014), 1409.1556.
[56] Vivienne Sze, et al. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE*, Vol. 105. 2295–2329.
[57] Tianqi Tang, et al. 2017. Binary convolutional neural network on RRAM. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC'17)*. 782–787.
[58] L. Wan, et al. 2013. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning (ICML'13)*. 105–1066.
[59] P. N. Whatmough, et al. 2017. A 28 nm SoC with a 1.2GHz 568nJ/prediction sparse deep-neural-network engine with 0.1 timing error rate tolerance for IoT applications. In *Proceedings of the International Solid-State Circuits Conference (ISSCC'17)*. 242–243.
[60] Benwei Xu, et al. 2016. A 23mW 24GS/s 6b time-interleaved hybrid two-step ADC in 28 nm CMOS. In *VLSI Circuits*. IEEE, 1–2.
[61] Xiaowei Xu, et al. 2017. Edge segmentation: Empowering mobile telemedicine with compressed cellular neural networks. In *Proceedings of the 36th International Conference on Computer-Aided Design*. 880–887.
[62] Y. Xu, et al. 2014. A 7-bit 40 MS/s single-ended asynchronous SAR ADC in 65 nm CMOS. *Analog Integr. Circ. Sign. Process.* 80, 349 (2014).
[63] Peng Yao, et al. 2017. Face classification using electronic synapses. *Nat. Commun.* 8 (2017), 15199.
[64] Shimeng Yu, et al. 2013. A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation. *Adv. Mater.* 25, 12 (2013), 1774–1779.
[65] Wei Zhao, et al. 2006. New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Trans. Electr. Dev.* 53, 11 (2006), 2816–2823.

Received July 2018; revised November 2018; accepted January 2019