Cellular Neural Network Friendly Convolutional Neural Networks – CNNs with CNNs

András Horváth†, Michael Hillmer*, Qiuwen Lou*, X. Sharon Hu*, and Michael Niemier*
†Pázmány Péter Catholic University, Budapest, Hungary, Email: horvath.andras@itk.ppke.hu
*Department of Computer Science and Engineering, University of Notre Dame
Notre Dame, IN 46556, USA, Email: {mhillmer, qlou, shu, mniemier}@nd.edu

***Abstract*—This paper discusses the development and evaluation of a Cellular Neural Network (CeNN) friendly deep learning network for solving the MNIST digit recognition problem. Prior work has shown that CeNNs leveraging emerging technologies such as tunnel transistors can improve energy or EDP of CeNNs, while simultaneously offering richer/more complex functionality. Important questions to address are what applications can benefit from CeNNs, and whether CeNNs can eventually outperform other alternatives at the application-level in terms of energy, performance, and *accuracy*. This paper begins to address these questions by using the MNIST problem as a case study.**

I. Introduction

As Moore's Law based device scaling and accompanying performance scaling trends slow, there is continued interest in new technologies and computational models to perform a given task faster and in a more energy efficient manner [1], [2]. Furthermore, there is growing evidence that, with respect to traditional Boolean circuits, it will be challenging for emerging devices to compete with CMOS technology [1]–[3].

Researchers are looking for new ways to exploit unique characteristics of emerging devices (e.g., non-volatility) as well as architectural paradigms that transcend the energy efficiency wall. In this regard, cellular neural networks (CeNNs) are now under investigation via the Semiconductor Research Corporation's benchmarking activities [3] as **(i)** they can solve a broad set of problems [4], and **(ii)** can leverage unique properties of both spin- and charge- based devices [2], [5].

Regardless of whether the CeNN (focus of this paper) or another emerging architecture/paradigm is studied, comparisons amongst said approaches should be made not just at the device level, but at the *application level* as well. Without proper application-level perspective, a new paradigm may appear promising – e.g., by simply considering core computational logic elements. However, without application-level perspective, important and necessary computational overheads may not be properly accounted for – e.g., digital-to-analog (D-to-A) and analog-to-digital (A-to-D) converters for analog processing elements, memory transfers, etc.

At present, many applications of interest involve various types of classification problems. With classification-type problems (which will also serve as a focus of this paper), we should not restrict ourselves to "traditional" metrics such as energy and delay. Of equal importance is the *accuracy* of the classification itself, especially as efforts to improve energy efficiency of inference and/or other applications frequently use computational accuracy – and hence classification accuracy – as a design knob.

As just two examples, [6] has considered limited precision analog hardware inter-mixed with conventional digital microprocessors. ("Approximable code" that can tolerate imprecise execution is mapped to analog hardware.) Preliminary studies suggest application-level speedups of 3.7X and energy savings of 6.3X with a quality loss of $< 10\%$ in most cases. Additionally, Shannon-inspired computing efforts [7] aim to generate useful results with circuits and systems composed of nano-scale, beyond-CMOS devices. While such devices offer the promise of improved energy efficiency when compared to CMOS counterparts, they are often intrinsically statistical when considering device operation. The impact of this device behavior must be accounted for at the application level.

Thus, while there are growing efforts to improve the energy efficiency, performance, and accuracy of inference applications, systematic studies to compare energy, delay, and *accuracy* tradeoffs for a common application (i.e., when different computational models, devices, etc. are employed) are limited. This paper aims to bring this problem to the forefront, and bring focus to the semi-disparate efforts such as those highlighted above. More specifically, we consider **(i)** the impact that CeNNs might have at the application level; as a case study, we use CeNNs to realize convolutional neural networks (CoNNs) that are becoming more ubiquitous and are relevant to numerous application spaces and problems [8], **(ii)** apply a CeNN-friendly CoNN to a standard inference problem – MNIST digit classification [9], and **(iii)** present initial accuracy/energy/delay projections for the CeNN approach – which is then compared with other projections for the same problem from the published literature.

II. Background and Related Work

A. Convolutional Neural Networks

As discussed in [8], a typical CoNN consists of a series of convolution, pooling, and non-linear activation stages. An input image may be fed into a sequence of convolution and pooling layers. A convolution layer is further broken down into different feature maps. Here, each unit is then connected to regions of feature maps of prior layers via filter banks (where a filter bank is a set of weights). The generated weighted sums are then processed via a non-linear operation – i.e., like the ReLU step to be discussed in Sec. III. Pooling units determine the maximum value from some group of units within some feature map and/or among feature maps themselves.

978-3-9815370-8-6/17/$31.00

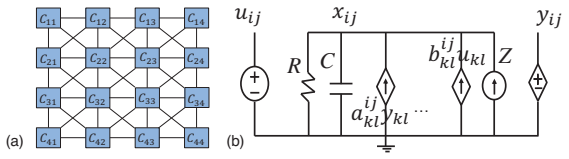

Figure 1. (a) CeNN architecture, (b) circuitry in CeNN cell.

B. Cellular Neural Networks

A spatially invariant CeNN architecture [4] is an $M \times N$ array of identical cells (Fig. 1a). Each cell, C_{ij}, $(i,j) \in \{1,...,M\} \times \{1,...,N\}$, has identical connections with adjacent cells in a predefined neighborhood, $N_r(i,j)$ of radius r. The size of the neighborhood is $m = (2r+1)^2$, where r is a positive integer. A conventional analog CeNN cell consists of one resistor, one capacitor, $2m$ linear voltage controlled current sources (VCCSs), one fixed current source, and one specific type of non-linear voltage controlled voltage source (Fig. 1b). The input, state, and output of a given cell C_{ij}, correspond to the nodal voltages, u_{ij}, x_{ij}, and y_{ij} respectively. VCCSs controlled by the input and output voltages of each neighbor deliver feedback and feedforward currents to a given cell. The dynamics of a CeNN are captured by a system of $M \times N$ ordinary differential equations, each of which is simply the Kirchhoff's Current Law (KCL) at the state nodes of the corresponding cells per Eq. 1.

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

CeNN cells typically employ a non-linear sigmoid-like transfer function [10] at the output to ensure fixed binary output levels.

The parameters $a_{ij,kl}$, and $b_{ij,kl}$ serve as weights for the feedback and feedforward currents from cell C_{kl} to cell C_{ij}. $a_{ij,kl}$, and $b_{ij,kl}$ are space invariant and are denoted by two $(2r+1) \times (2r+1)$ matrices. (If $r = 1$, they are captured by 3×3 matrices.) The matrices of a and b parameters are typically referred to as the feedback template (A) and the feedforward template (B) respectively. Design flexibility is further enhanced by the fixed bias current Z that provides a means to adjust total current flowing into a cell. A CeNN can solve a wide range of image processing problems by carefully selecting the values of the A and B templates (as well as Z).

Various circuits including inverters, Gilbert multipliers, operational transconductance amplifiers (OTAs), etc. [11], [12] can be used to realize VCCSs. For work to be discussed in this paper, we use the OTA design from [5]. OTAs provide a large linear range for voltage to current conversion, and can implement a wide range of transconductances allowing for different CeNN templates. Per Sec. III and IV, *non-linear* templates/OTAs can lead to CeNNs with richer functionality.

III. CeNN-friendly MNIST – Algorithms

At the highest levels, both CoNNs and CeNNs are capable of replicating certain aspects of human vision – which is arguably superior to any artificial platform when considering complexity, accuracy, and power consumption. In more detail, the origins of CoNNs can be traced back to the Neocognitron [13], where many aspects of the human vision system were modeled. There is a direct analogy between **(a)** the alternating simple/complex cells of the V1 area of the primary visual cortex and the spatial response of these cells, and **(b)** alternating layers of pooling and convolution operations. Thus, in an effort to mimic human vision, CoNNs **(i)** employ a grid-based structure of similar processing elements, **(ii)** process information topologically, and **(iii)** employ hardware that can efficiently perform a larger number of convolution operations.

As **(a)** the above three criteria also apply to CeNNs, **(b)** CeNNs operate in the analog domain – which could result in lower power consumption/improved energy efficiency [14], and **(c)** CeNNs are Turing complete [15] and could provide a richer library of functionality than that which is typically associated with CoNNs, here we consider how the topographic, highly parallel CeNN architecture can efficiently implement deep-learning operations/CoNNs.

CeNNs are typically comprised of a single layer of processing elements (PEs). Thus, while most CeNN hardware implementations lack the layered structure of CoNNs, by using local memory (commonly available on every realized CeNN chip), a cascade of said operations can be realized by re-using the result of each previous processing layer [16]. Alternatively, one could also simply build a cascade of CeNNs to realize the layered structure of a CoNN. This lends itself well to deep learning algorithms, which are built as cascades of different layers of non-linear processing elements, where every layer implements a simple operation that might include: **(i)** convolution, **(ii)** non-linear operations (usually a rectifier), **(iii)** pooling operations, and **(iv)** fully connected layers (although sometimes support vector machines (SVMs) are used instead).

Below, we describe how each of these steps 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 [17] [18], which is also briefly discussed at the end of this section.

A. Convolution

Convolution layers are used to detect and extract different feature maps on input data as the summation of the point-wise multiplication of two feature maps. One map is the input image f, and the other map 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 – the key element in deep learning architectures – can be defined per Eq. 2.

$$f * g(i,j) = \sum_{k,l=-\infty}^{\infty} f(i-k, j-l)g(k,l) \quad (2)$$

In deep learning architectures, convolution serves as a simple change detector. These are the only layers with parameters that are changed online, and the exact convolutional kernels are optimized during training.

As can be seen from Eq. 1 with the application of the feed-forward template (denoted as $b_{ij,kl}$), CeNNs can implement convolutional kernels in a straightforward manner.

B. 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 [19] is the rectified linear unit (ReLU) that per Eq. 3, thresholds every value below zero.

$$R(x) = \begin{Bmatrix} 0, if x \leq 0 \\ x, if x > 0 \end{Bmatrix} \tag{3}$$

In a CeNN, the ReLU operation can be implemented using a non-linear template (usually noted as $\hat{D}$ in the literature). Thus, to realize non-linear feedback, one can define an additional template implementing the non-linear function of the ReLU operation: $D(x_{i,j}) = \max(0, x_{i,j})$. This function sets all negative values to zero and leaves the positive values unchanged, because it directly implements Eq. 3. 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, non-linear operations prove to be much more difficult to realize. When considering actual hardware implementations, to date, real devices such as the Ace16k chip [20] or the SPS 02 Smart Photosensor from Toshiba [21] apply the standard CeNN non-linearity (see Eq. 4) on the cells.

$$y_{k,l} = \frac{1}{2}\left|x_{k,l}+1\right| - \frac{1}{2}\left|x_{k,l}-1\right| \tag{4}$$

Recall that the CeNN-UM is Turing complete, and it was proven by Chua and Roska [22] that all non-linear templates can be implemented as a series of linear templates using the standard CeNN non-linearity. For example, the ReLU operation can also be rewritten as a series of linear operations by applying templates as described below.

First one can execute the feed-forward template given by Eq. 5, which will simply decrease 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 Eq. 6:

$$B_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, Z = 1 \tag{6}$$

As the non-linearity thresholds a given value, these two linear operations implement the required ReLU operator – which leaves all positive values unchanged, and thresholds all values below 0. For completeness, we should also note that the standard CeNN non-linearity thresholds all elements above one as well. However, this is not problematic as we know the possible values of both the input elements and the convolution kernels. Thus, it can be ensured that values from the convolutional layers cannot be larger than 1.

In addition to using multiple linear templates to realize a non-linear template, unique OTA designs can actually implement certain non-linear templates directly. This should lead to higher performance and energy saving. More details on this will be given in Section IV.

C. Pooling

Pooling operations are employed to decrease the amount of information between consecutive layers in a deep neural network to compensate for the effects of small translations. A pooling operation selects the maximum element in a region around every value – i.e., per Eq. 7.

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

The pooling operation can also be implemented by the non-linear, GLOBMAX template, which can be found in the standard CeNN template library [23]. 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. This nonlinear template can also be implemented by using the $\hat{D}$ type non-linear function as given in Eq. 8.

$$D(x_{i,j}) = \begin{Bmatrix} -\frac{1}{8}x, \text{ if } x \leq 0 \\ 0, \text{ if } x > 0 \end{Bmatrix} \tag{8}$$

As before, desired functionality typically associated with non-linear template operations can also be realized with a sequence of *linear* operations. In more detail, consider the linear shift down template in Eq. 9:

$$B_1 = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{9}$$

This simple template shifts an input image down by one pixel and multiplies the pixels value by -1. After this step, by using an analog accumulator, we can subtract this shifted image from the original image. We then check all the values on the subtracted image and threshold the values at zero, leaving only the positive differences. (This thresholding operation is the same as the ReLU operation and can also leverage the non-linear OTA design to be discussed in Section IV.) By using the accumulator we can add this positive difference to the original image. This algorithm ensures that if the pixel above our input pixel is larger, our input pixel is increased and the resulting pixel has the same value as the previous maximum value. If we repeat this operation in all four directions via rotating the B template by 90 degrees at each step, we achieve the functionality associated with the non-linear GLOBMAX template, i.e., propagating the maximum value through all local neighbors in a CeNN array.

D. Fully-Connected Layers

The operations described in the above three sections are used as 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, the most common approach is to employ a fully connected layer and associated neurons. The fully connected layer considers information globally and unifies local features from the lower layers. It can be defined as a dot product between a weight map and the input data, or as a large convolution between the two maps. This product can be used a weight, which represents how strongly the data belongs into a class and the product is calculated for every class independently. The index of the largest weight can be selected and associated with the input data. The multiplication of the feature map and weight map – i.e., the point-wise calculation – cannot be efficiently implemented with CeNN template operations, and would likely be performed by a digital processor or analog dot-product engine (e.g., per [24]).

E. A CeNN-based CoNN for MNIST

Using the building blocks described above, we have developed a CeNN-friendly structure for the MNIST problem. In the MNIST handwritten digit classification task [9], a system must analyze and classify what digit (0-9) is represented by a 28×28 pixel black and white image. There are 60,000 images in the training set, and 10,000 images in the test set. Per the discussion above, 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 Eq. 1, all values would simply be 0) As such, the training of the network can also be done with a CeNN with backpropagation methods such as stochastic gradient descent [25].

Additionally, all computational kernels are 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 consistent with CeNN theory (i.e., per Sec. II, 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, this is not necessarily a restriction. Larger kernels can be estimated by using a series of 3×3 kernels, and it is common practice to substitute larger kernels with 3×3 operations. Furthermore, recent work [26] suggests that smaller kernels can lead to fewer parameters/higher accuracy during training. Again, this maps well to CeNN hardware.

Given the above, we have created a simulation framework based on existing CoNN frameworks – specifically the ConvNet [27] structure. (We have also implemented a more versatile/adjustable training framework in MATLAB – see [28].) Our network learns the parameters of the B-type templates for the convolution kernels. (Per the above, the B-template values for the ReLU and pooling layers are fixed.) A resulting layered, CeNN-friendly network is shown in Fig. 2. Note that if only linear templates are employed, 68 template operations are required. If non-linear template operations can ultimately be employed, this number could be reduced to 44.

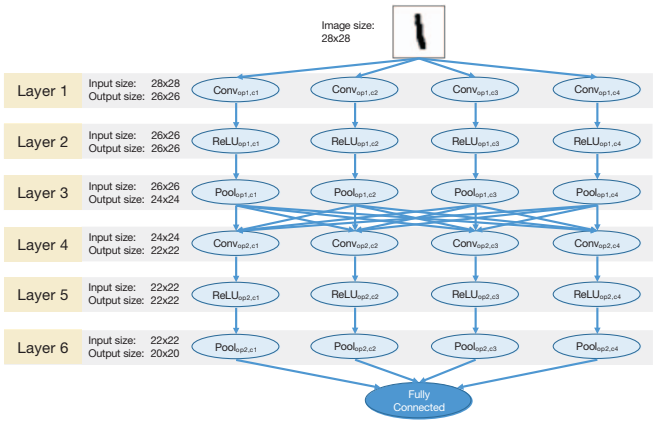

Figure 2. CeNN-friendly CoNN.

IV. CeNN-friendly MNIST – Accuracy and Hardware

In this section, we explore the "competitiveness" of a CeNN friendly CoNN when compared to other efforts discussed in the literature. We begin by considering classification accuracy which, per the discussion in Sec. V, should be at least on the order of 90% to 95%.

A. Numerical Simulations of CeNN-based CoNNs

Using a subset of test images from the MNIST dataset, initial numerical simulation results – with 32-bit precision – suggest that our CeNN-friendly CoNN has a classification accuracy of 97%, which is competitive with other approaches in the published literature.

That said, CeNNs are analog in nature, and it is not reasonable to expect 32-bit digital precision from CeNN hardware. As initial studies of the effects of reduced precision on classification accuracy, we repeated the aforementioned simulations with reduced binary precision. Per Table I, even with just 4-bit precision (i.e., to represent template values for convolution operations) classification accuracy is still $> 96\%$. (This is calculated by using a set of 10,000 test images.)

Table I
Accuracy with respect to the number of bits used

Number of Bits	Accuracy
32	97%
6-to-8	97%
5	96%
4	96%
3	86%
2	20%
1	10%

B. Numerical Simulations of CeNN-based CoNNs with Device Specific I-Vs

To further study the impact of analog CeNN hardware on classification accuracy, we consider in more detail the circuit elements that would actually be used to implement template values. As noted in Sec. III, both linear and non-linear template operations can be employed to realize CeNN layer functionality. In practice (hardware), **linear** template values are realized with OTAs. Leveraging prior work from [5], we consider the impact of a (TFET-based) OTA design (Fig. 3a) where a source degeneration resistor is used to enhance the linearity of the output. The actual template value is simply $G_m \cdot R$. (R is the in-cell resistor; as R is fixed, the G_m of the OTA is used to specify/realize a convolution template value per the discussion in Sec. III.) By tuning the G_m of the OTA, we can realize different template values.

Prior work has also considered how circuits/devices might be employed to realize hardware to more readily implement **non-linear** templates. For example, in [5], a new OTA design was proposed to realize a non-linear template. The primary focus of [5] was *not* to realize template values to support CoNNs, but to help reduce the number of template operations/steps in a CeNN algorithm for an arbitrary problem. That said, the hardware described in [5] is quite relevant for CoNNs.

As an example, per Sec. III-C, a pooling operation could be realized by a GLOBMAX template operation, and the modified OTA in Fig. 3b provides the necessary non-linearity. A TFET device is used between the source of two input transistors to realize the desired non-linear characteristics. When $V_{in+} > V_{in-}$, the I-V characteristic is linear – and G_m can be tuned to realize a desired (linear) template value. When $V_{in+} < V_{in-}$, in the ideal case G_m (and hence the output current) is 0. This design should also be extensible to the non-linear ReLU templates. By simply switching the two input terminals of the OTAs, one can readily realize the function described in Eq. 3.

While the above discussion suggests that the necessary characteristic for both linear and non-linear template operations for CeNN-based CoNNs can be realized with a single OTA, we must also account for any deviations associated with a circuit realization from the ideal function. In more detail, the I-V characteristics of a linear OTA are shown in Fig. 4a. For the circuit simulation, a GaSb/InAs TFET [29] device model is assumed. The G_m of the OTA can be estimated by Eq. 10,

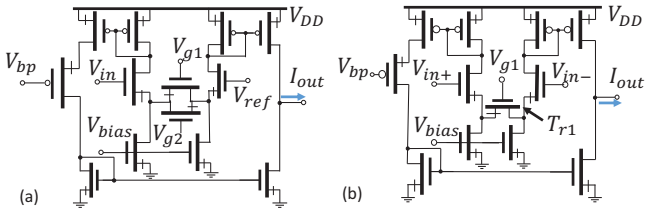

Figure 3. (a) TFET-based linear circuit, (b) TFET-based nonlinear circuit.

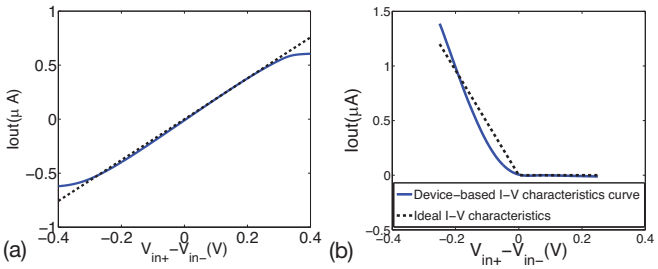

Figure 4. Ideal and actual characteristics associated with (a) linear and (b) non-linear OTAs.

where g_m is the transconductance of the input transistors.

$$G_m = \frac{g_m}{1 + g_m \cdot R_s} \tag{10}$$

The I-V characteristics of a non-linear OTA are given in Fig. 4b. Clearly, there is deviation from the ideal. (See [5] for more discussions regarding output error.)

To quantify how hardware discrepancies between ideal / device-based characteristics impact classification accuracy, we modified our MATLAB-based CeNN simulator to model the I-V characteristics of the OTAs per Fig. 4 (i.e., the adjusted template value reflects $G_m \cdot R$, which is more representative of what an actual circuit may generate). Based on this modified simulator, we considered classification accuracy for a CeNN-based CoNN. (In these initial studies, we have only considered linear template operations.) After simulating 100 test images (due to simulation time constraints) from the MNIST dataset, our results suggests that accuracy drops from 97% to 95%, and is still promising in the context of accuracy metrics. It is also worthwhile to point out that accuracy could be improved further if the non-ideal templates are used in the training process (to be done in our future work).

V. Initial Evaluation and Discussion

As initial accuracy results appear promising, we also consider initial projections for energy and delay of the CeNN-based CoNN. We designed a hardware accelerator as a co-processor to evaluate energy and delay. The hardware accelerator contains four CeNNs. Each CeNN is connected to an analog memory array of the same size, read and writes occur in parallel, and operations per Fig. 2 can proceed in parallel. For analog memories, we use the design from [30]. We used Hspice to simulate the analog memory schematics and to determine read/write delay. While a detailed discussion is beyond the scope of this paper, note that our simulations also account for additional overheads – e.g., after the convolution step, we perform an accumulation function by using CeNNs as an analog adder by tuning the templates in Eq. 11.

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, Z = 0 \tag{11}$$

Similarly, with pooling, an accumulator is needed as well, and we again recursively use the CeNN as an adder. Note that for the feature extraction layers (layers 1-6 in Fig. 2), computation

Table II
ACCURACY, ENERGY, AND DELAY FOR MNIST CLASSIFICATION

Approach	Misclass. Rate (%)	Energy/ class. (μJ)	Time/ class. (μ s)	AED
CeNN$_L$	5	1.77	634	$\sim$ 5,600
CeNN$_{NL}$	N/A	0.93	262	N/A
TN [31]	7.3	0.27	1000	$\sim$ 2,000
TN [31]	$\sim$ 5	$\sim$ 5	1000	$\sim$ 25,000
TN [31]	$\sim$ 3	$\sim$ 8	1000	$\sim$ 22,000
TN [31]	$\sim$ 0.9	$\sim$ 28	1000	$\sim$ 25,000
TN [31]	0.58	108	1000	$\sim$ 63,000
DC [32]	0.21	1,082	7.3	$\sim$ 1,700

takes place entirely in the analog domain – eliminating any A-to-D overheads. Only in the last step is a conversion required.

Initial energy/delay projections appear in Table II. These projections do not yet account for the overhead of the fully connected layer. That said, numerical simulations / analysis of the CoNN indicate that $\sim 90\%$ of computation is done in the feature extraction layers, and $\sim 10\%$ is associated with the fully connected layer. As such, layers 1-6 of the network in Fig. 2 are expected to dominate computational costs. Also, for a more complex network (for solving other non-MNIST problems) this ratio should continue to grow.

Finally, classification accuracy as well as energy/delay per classification for other approaches to solving the MNIST problem are summarized in Table II. TrueNorth (TN) data (28 nm) is extracted from [31], while DropConnect (DC) [32] data were collected via our own measurements on a CPU/GPU platform (Intel i5, 14 nm/NVIDIA Tesla, 28 nm). For each approach – which assumes hardware with similar feature sizes to our 20 nm device model – we report misclassification rates, energy/classification, time/classification, and an accuracy $\times$ energy $\times$ delay benchmark (lower numbers are desirable). When compared to existing approaches, the CeNN-based CoNN is on par with other approaches (i.e., in AED), and additional improvements (as discussed above and below) are possible.

In future work, we will quantify overhead of the fully-connected layer assuming both analog (e.g., [24]) and digital dot product engines. Additionally, we will consider AED metrics assuming non-linear ReLU and pooling templates. Moreover, we will consider CeNN-based training (including actual OTA I-V characteristics) to further improve accuracy. More generally, CeNNs should be explored further in the context of CoNNs for other problems.

ACKNOWLEDGMENT

This work was supported in part by the Center for Low Energy Systems Technology (LEAST), one of six SRC STARnet centers sponsored by MARCO and DARPA.

REFERENCES

[1] D. Nikonov and I. Young, "Benchmarking of beyond-cmos exploratory devices for logic integrated circuits," *IEEE J. on Exploratory Solid-State Computational Devices and Circuits*, vol. 1, pp. 3–11, Dec 2015.

[2] C. Pan and A. Naeemi, "A proposal for energy-efficient cellular neural network based on spintronic devices," *IEEE Transactions on Nanotechnology*, vol. 15, no. 5, pp. 820–827, Sept 2016.

[3] SRC, "SRC benchmarking center," 2016. [Online]. Available: https://www.src.org/program/nri/benchmarking/

[4] L. O. Chua and T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Applications*. New York, NY, USA: Cambridge University Press, 2002.

[5] Qiuwen Lou, *et al*, "Tfet-based operational transconductance amplifier design for CNN systems," in *ACM Great Lakes Symposium on VLSI*, 2015, pp. 277–282.

[6] R. St. Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," in *41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14, 2014, pp. 505–516.

[7] "Systems on nanoscale information fabrics (SONIC)," https://www.sonic-center.org/research/nano.php.

[8] Y. LeCun *et al.*, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[9] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[10] L O Chua and Lin Yang, "Cellular neural network: Theory," *IEEE Transactions on Circuits and Systems*, vol. 35, pp. 1257–1272, 1988.

[11] Jesus E. Molinar-Solis, *et al*, "Programmable cmos cnn cell based on floating-gate inverter unit," in *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 2007, pp. 207–216.

[12] Lei Wang, *et. al*, "Time multiplexed color image processing based on a CNN with cell-state outputs," *IEEE TVLSI*, vol. 6.2, pp. 314–322, 1998.

[13] F. Kunihiko, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36.4, pp. 193–202, 1980.

[14] K. Kim *et al.*, "A 125 gops 583 mw network-on-chip based parallel processor with bio-inspired visual attention engine," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 136–147, 2009.

[15] L. O. Chua and T. Roska, *Cellular neural networks and visual computing: foundations and applications*. Cambridge University Press, 2002.

[16] ——, *Cellular neural networks and visual computing: foundations and applications*, 2002.

[17] Y. LeCun *et al.*, "Deep learning," *Nature*, vol. 521.7553, pp. 436–444, 2015.

[18] C. S. et al, "Going deeper with convolutions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[19] G. E. Dahl *et al.*, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8609–8613.

[20] A. Rodríguez-Vázquez *et al.*, "Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 5, pp. 851–863, 2004.

[21] "Official site of the toshiba sps 02 smart photosensor," http://www.toshiba-teli.co.jp/en/products/industrial/sps/sps.htm.

[22] T. Roska and L. O. Chua, "The cnn universal machine: an analogic array computer," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, no. 3, pp. 163–173, 1993.

[23] "Software library for cellular wave computing engines in an era of kilo-processor chips version 3.1," http://cnn-technology.itk.ppke.hu/Template_library_v3.1.pdf, accessed: 2016-11-29.

[24] I. Nahlus *et al.*, "Energy-efficient dot product computation using a switched analog circuit architecture," in *International Symposium on Low Power Electronics and Design (ISLPED)*, Aug 2014, pp. 315–318.

[25] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[26] C. Szegedy *et al.*, "Rethinking the inception architecture for computer vision," *arXiv preprint arXiv:1512.00567*, 2015.

[27] K. Chatfield *et al.*, "Efficient on-the-fly category retrieval using Convnets and GPUs," in *Asian Conf. on Computer Vision*, 2014, pp. 129–145.

[28] "Pázmány péter catholic university," http://users.itk.ppke.hu/~horan/Caffe/convnetjs/convnet.html.

[29] G. Zhou *et al.*, "Novel gate-recessed vertical inas/gasb tfets with record high ion of 180 ua/um at vds 0.5 v," in *2012 International Electron Devices Meeting*, 2012, pp. 32.6.1–32.6.4.

[30] R. Carmona-Galan *et al.*, "An 0.5- mu;m cmos analog random access memory chip for teraops speed multimedia video processing," *IEEE Transactions on Multimedia*, vol. 1, no. 2, pp. 121–135, 1999.

[31] S. Esser *et al.*, "Backpropagation for energy-efficient neuromorphic computing," *Neural Info. Proc. Systems (NIPS)*, pp. 1–9, 2015.

[32] L. Wan *et al.*, "Regularization of neural networks using dropconnect," *Proceedings of the 30th international conference on machine learning (iCML-13)*, pp. 105–1066, 2013.