2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture

ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars

Ali Shafiee*, Anirban Nag*, Naveen Muralimanohar†, Rajeev Balasubramonian*,
John Paul Strachan†, Miao Hu†, R. Stanley Williams†, Vivek Srikumar*
*School of Computing, University of Utah, Salt Lake City, Utah, USA
Email: {shafiee, anirban, rajeev, svivek}@cs.utah.edu
†Hewlett Packard Labs, Palo Alto, California, USA
Email: {naveen.muralimanohar, john-paul.strachan, miao.hu, stan.williams}@hpe.com

***Abstract*—**

A number of recent efforts have attempted to design accelerators for popular machine learning algorithms, such as those involving convolutional and deep neural networks (CNNs and DNNs). These algorithms typically involve a large number of multiply-accumulate (dot-product) operations. A recent project, DaDianNao, adopts a near data processing approach, where a specialized neural functional unit performs all the digital arithmetic operations and receives input weights from adjacent eDRAM banks.

This work explores an in-situ processing approach, where memristor crossbar arrays not only store input weights, but are also used to perform dot-product operations in an analog manner. While the use of crossbar memory as an analog dot-product engine is well known, no prior work has designed or characterized a full-fledged accelerator based on crossbars. In particular, our work makes the following contributions: (i) We design a pipelined architecture, with some crossbars dedicated for each neural network layer, and eDRAM buffers that aggregate data between pipeline stages. (ii) We define new data encoding techniques that are amenable to analog computations and that can reduce the high overheads of analog-to-digital conversion (ADC). (iii) We define the many supporting digital components required in an analog CNN accelerator and carry out a design space exploration to identify the best balance of memristor storage/compute, ADCs, and eDRAM storage on a chip. On a suite of CNN and DNN workloads, the proposed ISAAC architecture yields improvements of 14.8×, 5.5×, and 7.5× in throughput, energy, and computational density (respectively), relative to the state-of-the-art DaDianNao architecture.

***Index Terms*—CNN, DNN, memristor, analog, neural, accelerator**

I. Introduction

Machine learning algorithms have recently grown in prominence – they are frequently employed for mobile applications, as well as for data analysis in back-end data centers. Architectures that are optimized for machine learning algorithms, e.g., convolutional neural networks (CNNs) and the more general deep neural networks (DNNs), can therefore have high impact. Machine learning algorithms are amenable to acceleration because of a high degree of compute parallelism. They are also challenging because of the sizes of datasets and the need to avoid the memory wall [8].

A recent project has taken significant strides in this direction – the DaDianNao architecture [9] manages the memory wall with an approach rooted in *near data processing*. A DaDianNao system employs a number of connected *chips (nodes)*, each made up of 16 *tiles*. A tile implements a *neural functional unit (NFU)* that has parallel digital arithmetic units; these units are fed with data from nearby SRAM buffers and eDRAM banks. The dominant data structures in CNNs and DNNs are the synaptic weight matrices that define each neuron layer. These are distributed across several eDRAM banks on multiple tiles/nodes. The computations involving these weights are *brought to the eDRAM banks* and executed on adjacent NFUs, thus achieving near data processing. This requires moving the outputs of the previous neuron layer to the relevant tiles so they can merge with co-located synaptic weights to produce the outputs of the current layer. The outputs are then routed to appropriate eDRAM banks so they can serve as inputs to the next layer. Most of the chip area is used to store synaptic weights in eDRAM. The number of NFUs are far smaller than the number of neurons in a layer. Therefore, the NFUs are shared by multiple neurons in time-multiplexed fashion.

Given the relative scarcity of NFUs, DaDianNao adopts the following approach to maximize performance. A single CNN layer is processed at a time. This processing is performed in parallel on all NFUs in the system. The outputs are collected in eDRAM banks. Once a layer is fully processed, DaDianNao moves on to the next layer, again parallelized across all NFUs in the system. Thus, an NFU is used sequentially by a number of neurons in one layer, followed by a number of neurons in the next layer, and so on. This form of context-switching at each NFU is achieved at relatively low cost by moving the appropriate inputs/weights from eDRAM banks into SRAM buffers that feed the NFUs.

The proposed ISAAC architecture differs from the DaDianNao architecture in several of these aspects. Prior work has already observed that crossbar arrays using resistive memory are effective at performing many dot-product operations in parallel [33], [43], [53], [71], [78]. Such a *dot-product engine* is analog in nature, essentially leveraging Kirchoff's Law to yield a bitline current that is a sum of products. However, these papers do not leverage crossbars to create a full-fledged architecture for CNNs, nor do they characterize the behavior of CNN benchmarks. As this paper shows, a full-fledged crossbar-based CNN accelerator must integrate several digital

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.12

and analog components, and overcome several challenges.

The potential success of crossbar-based accelerators is also facilitated by the recent evolution of machine learning algorithms. The best image-analysis algorithms of 2012-2014 [34], [69], [80] have a few normalization layers that cannot be easily adapted to crossbars. An accelerator for those algorithms would require a mix of analog crossbars and digital NFUs. However, the best algorithms of the past year [24], [63] have shown that some of these problematic normalization layers are not necessary, thus paving the way for analog crossbar-based accelerators that are both efficient and accurate.

The overall ISAAC design is as follows. Similar to DaDianNao, the system is organized into multiple nodes/tiles, with memristor crossbar arrays forming the heart of each tile. The crossbar not only stores the synaptic weights, it also performs the dot-product computations. This is therefore an example of *in-situ computing* [2], [30], [53]. While DaDianNao executes multiple layers and multiple neurons on a single NFU with time multiplexing, a crossbar can't be efficiently re-programmed on the fly. Therefore, a crossbar is dedicated to process a set of neurons in a given CNN layer. The outputs of that layer are fed to other crossbars that are dedicated to process the next CNN layer, and so on. Such a design is easily amenable to pipelining. As soon as enough outputs are generated by a layer and aggregated in an eDRAM buffer, the next layer can start its operations. By designing such a pipeline, the buffering requirements between layers is reduced. This allows us to dedicate most of the chip real estate for dot product engines. To improve the throughput of a bottleneck layer and create a more balanced pipeline, more crossbars can be employed for that layer by replicating the weights. We also define the many digital components that must support the crossbar's analog computation. *This efficient pipeline is the first key contribution of the paper.*

We observe that the key overheads in a crossbar are the analog-to-digital converter (ADC) and digital-to-analog converter (DAC). *A second contribution of the paper is a novel approach to lay out the bits and perform the arithmetic so that these overheads are significantly lowered.*

Finally, *we carry out a design space exploration to identify how the architecture can be optimized for various metrics, while balancing the chip area dedicated to storage/compute, buffers, and ADCs.* For state-of-the-art full-fledged CNN and DNN applications, ISAAC improves upon DaDianNao by 14.8×, 5.5×, and 7.5× in throughput, energy, and computational density (respectively).

II. Background

A. CNNs and DNNs

Deep neural networks (DNNs) are a broad class of classifiers consisting of cascading layers of neural networks. Convolutional neural networks (CNNs) are deep neural networks primarily seen in the context of computer vision, and consist of four different types of layers: convolutional, classifier, pooling, and local response/contrast normalization (LRN/LCN). Of these, the convolutional and classifier are the two most important layers – these are the two layers primarily targeted by the ISAAC architecture because they involve dot product computations. The pooling layer typically involves a simple maximum or average operation on a small set of input numbers. The LRN layer is harder to integrate into the ISAAC design, and will be discussed in Section II-B.

A typical algorithm in the image processing domain starts with multiple convolutional layers that first extract basic feature maps, followed by more complex feature maps. Pooling layers are interleaved to reduce the sizes of feature maps; these layers typically use a maximum or average operation to map multiple neighboring neurons in the input to one output neuron. The normalization layers (LRN and LCN) mix multiple feature maps into one. Ultimately, classifier layers correlate the feature maps to objects seen during training.

The convolutional layer is defined in terms of a collection of kernels or filters, denoted by K below. Each kernel is defined by a 3D array of size $N_i \times K_x \times K_y$ and converts an input feature map (presented as a set of N_i matrices) into an output matrix or feature map. We refer to the number of such filters and output matrices in the layer by N_o. The (x, y) element of the k^{th} output matrix is defined as:

$$f_k^{out}(x,y) = \sigma(\sum_{j=0}^{N_i-1}\sum_{s=0}^{K_x-1}\sum_{t=0}^{K_y-1} f_j^{in}(x+s, y+t) \times K_{(k,x,y)}(j,s,t))$$

where $f_j^{in}(x, y)$ is the neuron at position (x, y) of input feature map j. In addition, $K_{(k,x,y)}(j, s, t)$ is the weight at position (j, s, t) of the k^{th} kernel. In many applications, $K_{(k,x,y)}(j, s, t)$ does not depend on (x, y), and is referred to as a shared kernel. Finally, $\sigma()$ is an activation function. We focus on a sigmoid function such as $tanh$, but it can also be easily adapted to model other variants, such as the recently successful ReLU.

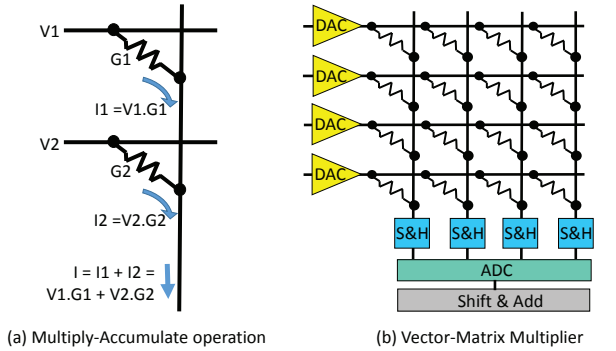
The classifier layer can be viewed as a special case of a convolution, with many output feature maps, each using the largest possible kernel size, i.e., a fully connected network.

In CNNs, the kernel weights in a convolutional layer are shared by all neurons of an output feature map, whereas DNNs use a private kernel for each neuron in an output feature map. DNN convolution layers with private kernels have much higher synaptic weight requirements.

B. Modern CNN/DNN Algorithms

We first summarize CNNs targeted at image detection and classification, such as the winners of the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [58]. These algorithms are judged by their top-5 error rates, i.e., how often is the target label not among the top 5 predictions made by the network. The winners in 2012 (AlexNet [34]), 2013 (Clarifai [80]), and 2014 (GoogLeNet [69]) achieved top-5 error rates of 15.3%, 11.2%, and 6.67% respectively. Based on the observations of Jarrett et al. [28], all of these networks incorporate LRN layers. LRN layers are not amenable to acceleration with crossbars, thus limiting the potential benefit of ISAAC for these algorithms.

However, results in the past year have shown that LRN is not as necessary as previously claimed. The Oxford VGG team showed [63] that on an improved version of AlexNet, not using LRN is slightly better than using LRN. They also built a network without LRN that has 16 layers and 138M parameters, and that achieves a top-5 error rate of 6.8% [63].



(a) Multiply-Accumulate operation (b) Vector-Matrix Multiplier

Fig. 1. (a) Using a bitline to perform an analog sum of products operation. (b) A memristor crossbar used as a vector-matrix multiplier.

Further, a Microsoft team [24] has built the best model to date with a top-5 error rate of 4.94% – this surpasses the human top-5 error rate of 5.1% [58]. The Microsoft network also does not include any LRN layers. It is comprised of three models: model A (178M parameters, 19 weight layers), model B (183M parameters, 22 weight layers) and model C (330M parameters, 22 weight layers).

We also examine the face detection problem. There are several competitive algorithms with LRNs [12], [37], [49], and without LRNs [67], [75]. DeepFace [75] achieves an accuracy of 97.35% on the Labeled Faces in the Wild (LFW) Dataset. It uses a Deep Neural Network with private kernels (120M parameters, 8 weight layers) and no LRN.

We use the above state-of-the-art CNNs and DNNs without LRN layers to compose our benchmark suite (Table II). We note that not every CNN/DNN operation can be easily integrated into every accelerator; it will therefore be important to engage in algorithm-hardware co-design [17].

C. The DaDianNao Architecture

A single DaDianNao [9] chip (node) is made up of 16 tiles and two central eDRAM banks connected by an on-chip fat-tree network. A tile is made up of a neural functional unit (NFU) and four eDRAM banks. An NFU has a pipeline with multiple parallel multipliers, a tree of adders, and a transfer function. These units typically operate on 16-bit inputs. For the transfer function, two logical units are deployed, each performing 16 piecewise interpolations (y=ax+b), the coefficients (a,b) of which are stored in two 16-entry SRAMs. The pipeline is fed with data from SRAM buffers within the tile. These buffers are themselves fed by eDRAM banks. Tiling is used to maximize data reuse and reduce transfers in/out of eDRAM banks. Synaptic weights are distributed across all nodes/tiles and feed their local NFUs. Neuron outputs are routed to eDRAM banks in the appropriate tiles. DaDianNao processes one layer at a time, distributing those computations across all tiles to maximize parallelism.

D. Memristor Dot Product Engines

Traditionally, memory arrays employ access transistors to isolate individual cells. Recently, resistive memories, especially those with non-linear IV curves, have been implemented with a crossbar architecture [74], [77]. As shown in Figure 1b, every bitline is connected to every wordline via resistive memory cells. Assume that the cells in the first column are programmed to resistances R_1, R_2,..., R_n. The conductances of these cells, G_1, G_2, ..., G_n, are the inverses of their resistances. If voltages V_1, V_2,...,V_n are applied to each of the n rows, cell i passes current V_i/R_i, or $V_i \times G_i$ into the bitline, based on Kirchoff's Law. As shown in Figure 1a, the total current emerging from the bitline is the sum of currents passed by each cell in the column. This current I represents the value of a dot product operation, where one vector is the set of input voltages at each row V and the second vector is the set of cell conductances G in a column, i.e., $I = V \times G$ (see Figure 1a).

The input voltages are applied to all the columns. The currents emerging from each bitline can therefore represent the outputs of neurons in multiple CNN output filters, where each neuron is fed the same inputs, but each neuron has a different set of synaptic weights (encoded as the conductances of cells in that column). The crossbar shown in Figure 1b achieves very high levels of parallelism – an $m \times n$ crossbar array performs dot products on m-entry vectors for n different neurons in a single step, i.e., it performs vector-matrix multiplication in a single step.

A sample-and-hold (S&H) circuit receives the bitline current and feeds it to a shared ADC unit (see Figure 1b). This conversion of analog currents to digital values is necessary before communicating the results to other digital units. Similarly, a DAC unit converts digital input values into appropriate voltage levels that are applied to each row.

The cells can be composed of any resistive memory technology. In this work, we choose memristor technology because it has an order of magnitude higher on/off ratio than PCM [35], thus affording higher bit density or precision. The crossbar is implemented with a 1T1R cell structure to facilitate more precise writes to memristor cells [79]. For the input voltages we are considering, i.e., DAC output voltage range, the presence of the access transistor per cell has no impact on the dot product computation.

III. Overall ISAAC Organization

We first present an overview of the ISAAC architecture, followed by detailed discussions of each novel feature. At a high level (Figure 2), an ISAAC chip is composed of a number of tiles (labeled T), connected with an on-chip concentrated-mesh (c-mesh). Each tile is composed of eDRAM buffers to store input values, a number of *in-situ multiply-accumulate (IMA)* units, and output registers to aggregate results, all connected with a shared bus. The tile also has shift-and-add, sigmoid, and max-pool units. Each IMA has a few crossbar arrays and ADCs, connected with a shared bus. The IMA also has input/output registers and shift-and-add units. A detailed discussion of each component is deferred until Section VI.

The architecture is not used for in-the-field training; it is only used for inference, which is the dominant operation in several domains (e.g., domains where training is performed once on a cluster of GPUs and those weights are deployed on millions of devices to perform billions of inferences). Adapting ISAAC for in-the-field training would require non-trivial effort and is left for future work.

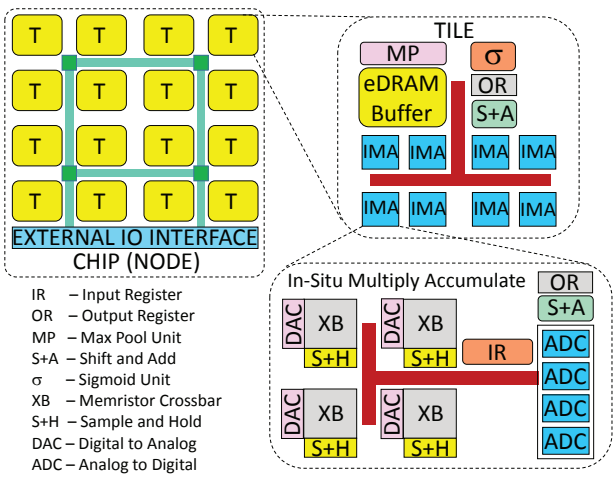


Fig. 2. ISAAC architecture hierarchy.

After training has determined the weights for every neuron, the weights are appropriately loaded into memristor cells with a programming step. Control vectors are also loaded into each tile to drive the finite state machines that steer inputs and outputs correctly after every cycle.

During inference, inputs are provided to ISAAC through an I/O interface and routed to the tiles implementing the first layer of the CNN. A finite state machine in the tile sends these inputs to appropriate IMAs. The dot-product operations involved in convolutional and classifier layers are performed on crossbar arrays; those results are sent to ADCs, and then aggregated in output registers after any necessary shift-and-adds. The aggregated result is then sent through the sigmoid operator and stored in the eDRAM banks of the tiles processing the next layer. The process continues until the final layer generates an output that is sent to the I/O interface. The I/O interface is also used to communicate with other ISAAC chips.

At a high level, ISAAC implements a hierarchy of chips/tiles/IMAs/arrays and c-mesh/bus. While the hierarchy is similar to that of DaDianNao, the internals of each tile and IMA are very different. A hierarchical topology enables high internal bandwidth, reduced data movement when aggregating results, short bitlines and wordlines in crossbars, and efficient resource partitioning across the many layers of a CNN.

IV. The ISAAC Pipeline

DaDianNao operates on one CNN layer at a time. All the NFUs in the system are leveraged to perform the required operations for one layer in parallel. The synaptic weights for that layer are therefore scattered across eDRAM banks in all tiles. The outputs are stored in eDRAM banks and serve as inputs when the next layer begins its operation. DaDianNao therefore maximizes throughput for one layer. This is possible because it is relatively easy for an NFU to context-switch from operating on one layer to operating on a different layer – it simply has to bring in a new set of weights from the eDRAM banks to its SRAM buffers.

On the other hand, ISAAC uses memristor arrays to not only store the synaptic weights, but also perform computations

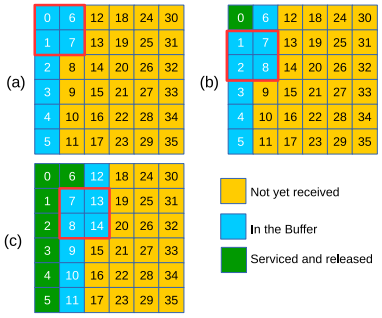


Fig. 3. Minimum input buffer requirement for a 6×6 input feature map with a 2×2 kernel and stride of 1. The blue values in (a), (b), and (c) represent the buffer contents for output neurons 0, 1, and 7, respectively.

on them. The in-situ computing approach requires that if an array has been assigned to store weights for a CNN layer, it has also been assigned to perform computations for that layer. Therefore, unlike DaDianNao, the tiles/IMAs of ISAAC have to be partitioned across the different CNN layers. For example, tiles 0-3 may be assigned to layer 0, tiles 4-11 may be assigned to layer 1, and so on. In this case, tiles 0-3 would store all weights for layer 0 and perform all layer 0 computations in parallel. The outputs of layer 0 are sent to some of tiles 4-11; once enough layer 0 outputs are buffered, tiles 4-11 perform the necessary layer 1 computations, and so on.

To understand how results are passed from one stage to the next, consider the following example, also shown in Figure 3. Assume that in layer i, a 6×6 input feature map is being convolved with a 2×2 kernel to produce an output feature map of the same size. Assume that a single column in an IMA has the four synaptic weights used by the 2×2 kernel. The previous layer $i-1$ produces outputs 0, 1, 2, ..., 6, 7, shown in blue in Figure 3a. All of these values are placed in the input buffer for layer i. At this point, we have enough information to start the operations for layer i. So inputs 0, 1, 6, 7 are fed to the IMA and they produce the first output for layer i. When the previous layer $i-1$ produces output 8, it gets placed in the input buffer for layer i. Value 0, shown in green in Figure 3b, is no longer required and can be removed from the input buffer. Thus, every new output produced by layer $i-1$ allows layer i to advance the kernel by one step and perform a new operation of its own. Figure 3c shows the state of the input buffer a few steps later. Note that a set of inputs is fed to N_{of} convolutional kernels to produce N_{of} output feature maps. Each of these kernels constitutes a different column in a crossbar and operates on a set of inputs in parallel.

We now discuss two important properties of this pipeline. The first pertains to buffering requirements in eDRAM between layers. The second pertains to synaptic weight storage in memristors to design a balanced pipeline. In our discussions, a *cycle* is the time required to perform one crossbar read operation, which for most of our analysis is 100 ns.

The eDRAM buffer requirement between two layers is fixed. In general terms, the size of the buffer is:

$$((N_x \times (K_y - 1)) + K_x) \times N_{if}$$

where N_x is the number of rows in the input feature map, K_y and K_x are the number of columns and rows in the kernel,

and N_{if} is the number of input feature maps involved in the convolution step. Without pipelining, the entire output ($N_x \times N_y \times N_{if}$) from layer $i-1$ would have to be stored before starting layer i. Thus, pipelining helps reduce the buffering requirement by approximately N_y/K_y.

The $K_x \times K_y$ kernel is moved by strides S_x and S_y after every step. If say $S_x = 2$ and $S_y = 1$, the previous layer $i-1$ has to produce two values before layer i can perform its next step. This can cause an unbalanced pipeline where the IMAs of layer $i-1$ are busy in every cycle, while the IMAs of layer i are busy in only every alternate cycle. To balance the pipeline, we double the resources allocated to layer $i-1$. In essence, the synaptic weights for layer $i-1$ are replicated in a different crossbar array so that two different input vectors can be processed in parallel to produce two output values in one cycle. Thus, to estimate the total synaptic storage requirement for a balanced pipeline, we work our way back from the last layer. If the last layer is expected to produce outputs in every cycle, it will need to store $K_x \times K_y \times N_{if} \times N_{of}$ synaptic weights. This term for layer i is referred to as W_i. If layer i is producing a single output in a cycle, the storage requirement for layer $i-1$ is $W_{i-1} \times S_{xi} \times S_{yi}$. Based on the values of S_x and S_y for each layer, the weights in early layers may be replicated several times. If the aggregate storage requirement exceeds the available storage on the chip by a factor of $2\times$, then the storage allocated to every layer (except the last) is reduced by $2\times$. The pipeline remains balanced and most IMAs are busy in every cycle; but the very last layer performs an operation and produces a result only in every alternate cycle.

A natural question arises: is such a pipelined approach useful in DaDianNao as well? If we can design a well-balanced pipeline and keep every NFU busy in every time step in DaDianNao, to a first order, the pipelined approach will match the throughput of the non-pipelined approach. The key here is that ISAAC needs pipelining to keep most IMAs busy most of the time, whereas DaDianNao is able to keep most NFUs busy most of the time without the need for pipelining.

V. Managing Bits, ADCs, and Signed Arithmetic

The Read/ADC Pipeline

The ADCs and DACs in every tile can represent a significant overhead. To reduce ADC overheads, we employ a few ADCs in one IMA, shared by multiple crossbars. We need enough ADCs per IMA to match the read throughput of the crossbars. For example, a 128×128 crossbar may produce 128 bitline currents every 100 ns (one cycle, which is the read latency for the crossbar array). To create a pipeline within the IMA, these bitline currents are latched in 128 sample-and-hold circuits [48]. In the next 100 ns cycle, these analog values in the sample-and-holds are fed sequentially to a single 1.28 giga-samples-per-second (GSps) ADC unit. Meanwhile, in a pipelined fashion, the crossbar begins its next read operation. Thus, 128 bitline currents are processed in 100 ns, before the next set of bitline currents are latched in the sample-and-hold circuits.

Input Voltages and DACs

Each row in a crossbar array needs an input voltage produced by a DAC. We'll assume that every row receives its input voltage from a dedicated n-bit DAC. Note that a 1-bit DAC is a trivial circuit (an inverter).

Next, we show how high precision and efficiency can be achieved, while limiting the size of the ADC and DAC. We target 16-bit fixed-point arithmetic, partially because prior work has shown that 16-bit arithmetic is sufficient for this class of machine learning algorithms [8], [21], and partially to perform an apples-to-apples comparison with DaDianNao.

To perform a 16-bit multiplication in every memristor cell, we would need a 16-bit DAC to provide the input voltage, 2^{16} resistance levels in each cell, and an ADC capable of handling well over 16 bits. It is clear that such a naive approach would have enormous overheads and be highly error-prone.

To address this, we first mandate that the input be provided as multiple sequential bits. Instead of a single voltage level that represents a 16-bit fixed-point number, we provide 16 voltage levels sequentially, where voltage level i is a 0/1 binary input representing bit i of the 16-bit input number. The first cycle of this iterative process multiplies-and-adds bit 1 of the inputs with the synaptic weights, and stores the result in an output register (after sending the sum of products through the ADC). The second cycle multiplies bit 2 of the inputs with the synaptic weights, shifts the result one place to the left, and adds it to the output register. The process continues until all 16 bits of the input have been handled in 16 cycles. This algorithm is similar to the classic multiplication algorithm used in modern multiplier units. In this example where the input is converted into 1-bit binary voltage levels, the IMA has to perform 16 sequential operations and we require a simple 1-bit DAC. We could also accept a v-bit input voltage, which would require $16/v$ sequential operations and a v-bit DAC. We later show that the optimal design point uses $v = 1$ and eliminates the need for an explicit DAC circuit.

One way to reduce the sequential 16-cycle delay is to replicate the synaptic weights on (say) two IMAs. Thus, one IMA can process the 8 most significant bits of the input, while the other IMA can process the 8 least significant bits of the input. The results are merged later after the appropriate shifting. This halves the latency for one 16-bit dot-product operation while requiring twice as much storage budget. In essence, if half the IMAs on a chip are not utilized, we can replicate all the weights and roughly double system throughput.

Synaptic Weights and ADCs

Having addressed the input values and the DACs, we now turn our attention to the synaptic weights and the ADCs. It is impractical to represent a 16-bit synaptic weight in a single memristor cell [26]. We therefore represent one 16-bit synaptic weight with $16/w$ w-bit cells located in the same row. For the rest of this discussion, we assume $w = 2$ because it emerges as a sweet spot in our design space exploration. When an input is provided, the cells in a column perform their sum of products operations. The results of adjacent columns must then be merged with the appropriate set of shifts and adds.

If the crossbar array has R rows, a single column is adding the results of R multiplications of v-bit inputs and w-bit synaptic weights. The number of bits in the resulting computation dictates the resolution A and size of the ADC.

The relationship is as follows:

$$A = log(R) + v + w, \quad if\ v > 1\ and\ w > 1 \tag{1}$$

$$A = log(R) + v + w - 1, \quad otherwise \tag{2}$$

Thus, the design of ISAAC involves a number of independent parameters (v, w, R, etc.) that impact overall throughput, power, and area in non-trivial ways.

Encoding to Reduce ADC Size

To further reduce the size of the ADC, we devise an encoding where every w-bit synaptic weight in a column is stored in its original form, or in its "flipped" form. The flipped form of w-bit weight W is represented as $\bar{W} = 2^w - 1 - W$. If the weights in a column are collectively large, i.e., with maximal inputs, the sum-of-products yields an MSB of 1, the weights are stored in their flipped form. This guarantees that the MSB of the sum-of-products will be 0. By guaranteeing an MSB of 0, the ADC size requirement is lowered by one bit.

The sum-of-products for a flipped column is converted to its actual value with the following equation:

$$\sum_{i=0}^{R-1} a_i \times \bar{W_i} = \sum_{i=0}^{R-1} a_i \times (2^w - 1 - W_i) = (2^w - 1) \sum_{i=0}^{R-1} a_i - \sum_{i=0}^{R-1} a_i \times W_i$$

where a_i refers to the i^{th} input value. The conversion requires us to compute the sum of the current input values a_i, which is done with one additional column per array, referred to as the *unit column*. During an IMA operation, the unit column produces the result $\sum_{i=0}^{R-1} a_i$. The results of any columns that have been stored in flipped form is subtracted from the results of the unit column. In addition, we need a bit per column to track if the column has original or flipped weights.

The encoding scheme can be leveraged either to reduce ADC resolution, increase cell density, or increase the rows per crossbar. At first glance, it may appear that a 1-bit reduction in ADC resolution is not a big deal. As we show later, because ADC power is a significant contributor to ISAAC power, and because some ADC overheads grow exponentially with resolution, the impact of this technique on overall ISAAC efficiency is very significant. In terms of overheads, the columns per array has grown from 128 to 129 and one additional shift-and-add has been introduced. Note that the shift-and-add circuits represent a small fraction of overall chip area and there is more than enough time in one cycle (100 ns) to perform a handful of shift-and-adds and update the output register. Our analysis considers these overheads.

Correctly Handling Signed Arithmetic

Synaptic weights in machine learning algorithms can be positive or negative. It is important to allow negative weights so the network can capture inhibitory effects of features [70]. We must therefore define bit representations that perform correct signed arithmetic operations, given that a memristor bitline can only add currents. There are a number of ways that signed numbers can be represented and we isolate the best approach below. This approach is also compatible with the encoding previously described. In fact, it leverages the unit column introduced earlier.

We assume that inputs to the crossbar arrays are provided with a 2's complement representation. For a 16-bit signed fixed-point input, a 1 in the i^{th} bit represents a quantity of 2^i ($0 \leq i < 15$) or -2^i ($i = 15$). Since we have already decided to provide the input one bit at a time, this is easily handled – the result of the last dot-product operation undergoes a shift-and-subtract instead of a shift-and-add.

Similarly, we could have considered a 2's complement representation for the synaptic weights in the crossbars too, and multiplied the quantity produced by the most significant bit by -2^{15}. But because each memristor cell stores two bits, it is difficult to isolate the contribution of the most significant bit. Therefore, for synaptic weights, we use a representation with a bias (similar to the exponent in the IEEE 754 floating-point standard). A 16-bit fixed-point weight between -2^{15} and $2^{15} - 1$ is represented by an unsigned 16-bit integer, and the conversion is performed by subtracting a bias of 2^{15}. Since the bitline output represents a sum of products with biased weights, the conversion back to a signed fixed-point value will require that the biases be subtracted, and we need to subtract as many biases as the 1s in the input. As discussed before, the unit column has already counted the number of 1s in the input. This count is multiplied by the bias of 2^{15} and subtracted from the end result. The subtraction due to the encoding scheme and the subtraction due to the bias combine to form a single subtraction from the end result.

In summary, this section has defined bit representations for the input values and the synaptic weights so that correct signed arithmetic can be performed, and the overheads of ADCs and DACs are kept in check. The proposed approach incurs additional shift-and-add operations (a minor overhead), 16-iteration IMA operations to process a 16-bit input, and 8 cells per synaptic weight. Our design space exploration shows that these choices best balance the involved trade-offs.

VI. Example and Intra-Tile Pipeline

Analog units can be ideal for specific functions. But to execute a full-fledged CNN/DNN, the flow of data has to be orchestrated with a number of digital components. This section describes the operations of these supporting digital components and how these operations are pipelined.

This is best explained with the example shown in Figure 4. In this example, layer i is performing a convolution with a 4×4 shared kernel. The layer receives 16 input filters and produces 32 output filters (see Figure 4a). These output filters are fed to layer $i + 1$ that performs a max-pool operation on every 2×2 grid. The 32 down-sized filters are then fed as input to layer $i + 2$. For this example, assume that kernel strides (S_x and S_y) are always 1. We assume that one IMA has four crossbar arrays, each with 128 rows and 128 columns.

Layer i performs a dot-product operation with a $4 \times 4 \times 16$ matrix, i.e., we need 256 multiply-add operations, or a crossbar with 256 rows. Since there are 32 output filters, 32 such operations are performed in parallel. Because each of these 32 operations is performed across 8 2-bit memristor cells in a row, we need a crossbar with 256 columns. Since the operation requires a logical crossbar array of size 256×256, it must be spread across 4 physical crossbar arrays of size 128×128.

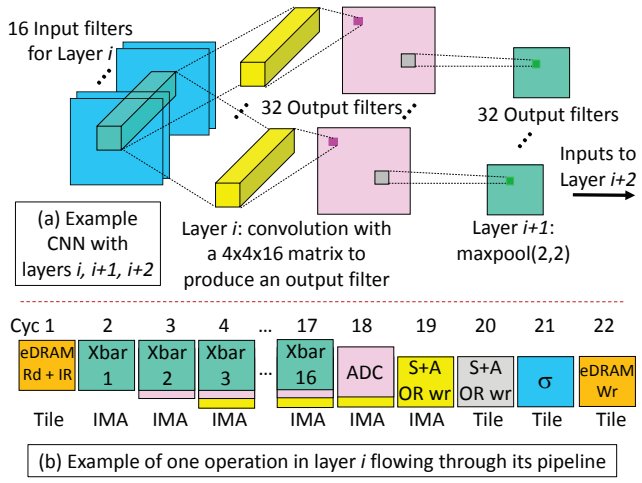


Fig. 4. Example CNN layer traversing the ISAAC pipeline.

A single IMA may be enough to perform the computations required by layer i.

The outputs of layer $i-1$ are stored in the eDRAM buffer for layer i's tile. As described in Figure 3, when a new set of inputs (N_i 16-bit values) shows up, it allows layer i to proceed with its next operation. This operation is itself pipelined (shown in detail in Figure 4b), with the cycle time (100 ns) dictated by the slowest stage, which is the crossbar read. In the first cycle, an eDRAM read is performed to read out 256 16-bit inputs. These values are sent over the shared bus to the IMA for layer i and recorded in the *input register (IR)*. The IR has a maximum capacity of 1KB and is implemented with SRAM. The entire copy of up to 1KB of data from eDRAM to IR is performed within a 100 ns stage. We design our eDRAM and shared bus to support this maximum bandwidth.

Once the input values have been copied to the IR, the IMA will be busy with the dot-product operation for the next 16+ cycles. In the next 16 cycles, the eDRAM is ready to receive other inputs and deal with other IMAs in the tile, i.e., it context-switches to handling other layers that might be sharing that tile while waiting for the result of one IMA.

Over the next 16 cycles, the IR feeds 1 bit at a time for each of the 256 input values to the crossbar arrays. The first 128 bits are sent to crossbars 0 and 1, and the next 128 bits are sent to crossbars 2 and 3. At the end of each 100 ns cycle, the outputs are latched in the Sample & Hold circuits. In the next cycle, these outputs are fed to the ADC units. The results of the ADCs are then fed to the shift-and-add units, where the results are merged with the output register (OR) in the IMA.

The OR is a 128B SRAM structure. In this example, it produces 32 16-bit values over a 16-cycle period. In each cycle, the results of the ADCs are shifted and added to the value in the OR (this includes the shift-and-adds required by the encoding schemes). Since we have 100 ns to update up to 64 16-bit values, we only need 4 parallel shift-and-add units, which represents a very small area overhead.

As shown in Figure 4b, at the end of cycle 19, the OR in the IMA has its final output value. This is sent over the shared bus to the central units in the tile. These values may undergo another step of shift-and-adds and merging with the central OR in the tile if the convolution is spread across multiple IMAs (not required for layer i in our example). The central OR contains the final results for neurons at the end of cycle 20. Note that in the meantime, the IMA for layer i has already begun processing its next inputs, so it is kept busy in every cycle.

The contents of the central OR are sent to the sigmoid unit in cycle 21. The sigmoid unit is identical to that used in DaDianNao, and incurs a relatively small area and power penalty. Finally, in cycle 22, the sigmoid results are written to the eDRAM that will provide the inputs for the next layer. In this case, since layer $i+1$ is being processed in the same tile, the same eDRAM buffer is used to store the sigmoid results. If the eDRAM had been busy processing some other layer in cycle 22, we would have to implement layer $i+1$ on a different tile to avoid the structural hazard.

To implement the max-pool in layer $i+1$ in our example, 4 values from a filter have to be converted into 1 value; this is repeated for 32 filters. This operation has to be performed only once every 64 cycles. In our example, we assume that the eDRAM is read in cycles 23-26 to perform the max-pool. The max-pool unit is made up of comparators and registers; a very simple max-pool unit can easily keep up with the eDRAM read bandwidth. The results of the max-pool are written to the eDRAM used for layer $i+2$ in cycle 27.

The mapping of layers to IMAs and the resulting pipeline have to be determined off-line and loaded into control registers that drive finite state machines. These state machines ensure that results are sent to appropriate destinations in every cycle. Data transfers over the c-mesh are also statically scheduled and guaranteed to not conflict with other data packets. If a single large CNN layer is spread across multiple tiles, some of the tiles will have to be designated as *Aggregators*, i.e., they aggregate the ORs of different tiles and then apply the Sigmoid.

VII. Methodology

Energy and Area Models

All ISAAC parameters and their power/area values are summarized in Table I. We use CACTI 6.5 [45] at 32 nm to model energy and area for all buffers and on-chip interconnects. The memristor crossbar array energy and area model is based on [26]. The energy and area for the shift-and-add circuits, the max-pool circuit, and the sigmoid operation are adapted from the analysis in DaDianNao [9]. For off-chip links, we employ the same HyperTransport serial link model as that used by DaDianNao [9].

For ADC energy and area, we use data from a recent survey [46] of ADC circuits published at major circuit conferences. For most of our analysis, we use an 8-bit ADC at 32 nm that is optimized for area. We also considered ADCs that are optimized for power, but the high area and low sampling rates of these designs made them impractical (450 KHz in 0.12 mm^2 [36]). An SAR ADC has four major components [36]: a vref buffer, memory, clock, and a capacitive DAC. To arrive at power and area for the same style ADC, but with different bit resolutions, we scaled the

power/area of the vref buffer, memory, and clock linearly, and the power/area of the capacitive DAC exponentially [59].

For most of the paper, we assume a simple 1-bit DAC because we need a DAC for every row in every memristor array. To explore the design space with multi-bit DACs, we use the power/area model in [59].

The energy and area estimates for DaDianNao are taken directly from that work [9], but scaled from 28 nm to 32 nm for an apples-to-apples comparison. Also, we assume a similar sized chip for both (an iso-area comparison) – our analysis in Table I shows that one ISAAC chip can accommodate 14×12 tiles. It is worth pointing out that DaDianNao's eDRAM model, based on state-of-the-art eDRAMs, yields higher area efficiency than CACTI's eDRAM model.

Performance Model

We have manually mapped each of our benchmark applications to the IMAs, tiles, and nodes in ISAAC. Similar to the example discussed in Section VI, we have made sure that the resulting pipeline between layers and within a layer does not have any structural hazards. Similarly, data exchange between tiles on the on-chip and off-chip network has also been statically routed without any conflicts. This gives us a deterministic execution model for ISAAC and the latency/throughput for a given CNN/DNN can be expressed with analytical equations. The performance for DaDianNao can also be similarly obtained. Note that CNNs/DNNs executing on these tiled accelerators do not exhibit any run-time dependences or control-flow, i.e., cycle-accurate simulations do not capture any phenomena not already captured by our analytical estimates.

Metrics

We consider three key metrics:

1) **CE:** Computational Efficiency is represented by the number of 16-bit operations performed per second per mm^2 ($GOPS/s \times mm^2$).
2) **PE:** Power Efficiency is represented by the number of 16-bit operations performed per watt ($GOPS/W$).
3) **SE:** Storage Efficiency is the on-chip capacity for synaptic weights per unit area (MB/mm^2).

We first compute the peak capabilities of the architectures for each of these metrics. We then compute these metrics for our benchmark CNNs/DNNs – based on how the CNN/DNN maps to the tiles/IMAs, some IMAs may be under-utilized.

Benchmarks

Based on the discussion in Section II-B, we use seven benchmark CNNs and two DNNs. Four of the CNNs are versions of the Oxford VGG [63], and three are versions of MSRA [24] – both are architectures proposed in ILSVRC 2014. DeepFace is a complete DNN [75] while the last workload is a large DNN layer [37] also used in the DaDianNao paper [9]. Table II lists the parameters for these CNNs and DNNs. The largest workload has 26 layers and 330 million parameters.

VIII. Results

A. Analyzing ISAAC

Design Space Exploration

ISAAC Tile at 1.2 GHz, 0.37 mm^2				
Component	**Params**	**Spec**	**Power**	**Area** (mm^2)
eDRAM Buffer	size num_banks bus_width	64KB 4 256 b	20.7 mW	0.083
eDRAM -to-IMA bus	num_wire	384	7 mW	0.090
Router	flit size num_port	32 8	42 mW	0.151 (shared by 4 tiles)
Sigmoid	number	2	0.52 mW	0.0006
S+A	number	1	0.05 mW	0.00006
MaxPool	number	1	0.4 mW	0.00024
OR	size	3 KB	1.68 mW	0.0032
Total			**40.9 mW**	**0.215** mm^2
IMA properties (12 IMAs per tile)				
ADC	resolution frequency number	8 bits 1.2 GSps 8	16 mW	0.0096
DAC	resolution number	1 bit 8×128	4 mW	0.00017
S+H	number	8×128	10 uW	0.00004
Memristor array	number size bits per cell	8 128×128 2	2.4 mW	0.0002
S+A	number	4	0.2 mW	0.00024
IR	size	2 KB	1.24 mW	0.0021
OR	size	256 B	0.23 mW	0.00077
IMA Total	number	12	**289 mW**	**0.157** mm^2
1 Tile Total			**330 mW**	**0.372** mm^2
168 Tile Total			**55.4 W**	**62.5** mm^2
Hyper Tr	links/freq link bw	4/1.6GHz 6.4 GB/s	10.4 W	22.88
Chip Total			**65.8 W**	**85.4** mm^2
DaDianNao at 606 MHz scaled up to 32nm				
eDRAM	size num_banks	36 MB 4 per tile	4.8 W	33.22
NFU	number	16	4.9 W	16.22
Global Bus	width	128 bit	13 mW	15.7
16 Tile Total			**9.7 W**	**65.1** mm^2
Hyper Tr	links/freq link bw	4/1.6GHz 6.4 GB/s	10.4 W	22.88
Chip Total			**20.1 W**	**88** mm^2

TABLE I
ISAAC Parameters.

ISAAC's behavior is a function of many parameters: (1) the size of the memristor crossbar array, (2) the number of crossbars in an IMA, (3) the number of ADCs in an IMA, and (4) the number of IMAs in a tile. The size of the central eDRAM buffer in a node is set to 64 KB and the c-mesh flit width is set to 32 bits. These were set to limit the search space and were determined based on the buffering/communication requirements for the largest layers in our benchmarks. Many of the other parameters, e.g., the resolution of the ADC, and the width of the bus connecting the eDRAM and the IMAs, are derived from the above parameters to maintain correctness and avoid structural hazards for the worst-case layers. This sub-section reports peak CE, PE, and SE values, assuming that all IMAs can be somehow utilized in every cycle.

Figure 5a plots the peak CE metric on the Y-axis as we sweep the ISAAC design space. The optimal design point has 8 128×128 arrays, 8 ADCs per IMA, and 12 IMAs per tile. We refer to this design as ISAAC-CE. The parameters, power, and area breakdowns shown in Table I are for ISAAC-CE. Figure 5b carries out a similar design space exploration with the PE metric. The configuration with optimal PE is referred to as ISAAC-PE. We observe that ISAAC-CE and ISAAC-

input size	VGG-1	VGG-2	VGG-3	VGG-4	MSRA-1	MSRA-2	MSRA-3	input size	DeepFace
224	3x3,64 (1)	3x3,64 (2)	3x3,64 (2)	3x3,64 (2)	7x7,96/2(1)	7x7,96/2(1)	7x7,96/2(1)	152	11x11,32(1)
	2x2 maxpool/2							142	3x3 maxpool/2
112	3x3,128 (1)	3x3,128 (2)	3x3,128 (2)	3x3,128 (2)				71	9x9,16/2(1)
	2x2 maxpool/2							63	9x9,16(1)*
56	3x3,256 (2)	3x3,256 (2) 1x1, 256(1)	3x3,256 (3)	3x3,256 (4)	3x3,256 (5)	3x3,256 (6)	3x3,384 (6)	55	7x7,16/2(1)*
	2x2 maxpool/2							25	5x5,16(1)*
28	3x3,512 (2)	3x3,512 (2) 1x1,256 (1)	3x3,512 (3)	3x3,512 (4)	3x3,512 (5)	3x3,512 (6)	3x3,768 (6)		FC-4096(1)
	2x2 maxpool/2								FC-4030(1)
14	3x3,512 (2)	3x3,512 (2) 1x1,512 (1)	3x3,512 (3)	3x3,512 (4)	3x3,512 (5)	3x3,512 (6)	3x3,896 (6)		
	2x2 maxpool/2				spp,7,3,2,1				
	FC-4096(2)								
	FC-1000(1)								
DNN: $N_x = N_y$=200, $K_x = K_y$=18, $N_o = N_i$=8									

TABLE II

BENCHMARK NAMES ARE IN BOLD. LAYERS ARE FORMATTED AS $K_x \times K_y, N_o$/STRIDE (T), WHERE T IS THE NUMBER OF SUCH LAYERS. STRIDE IS 1 UNLESS EXPLICITLY MENTIONED. LAYER* DENOTES CONVOLUTION LAYER WITH PRIVATE KERNELS.

PE are quite similar, i.e., the same design has near-maximum throughput and energy efficiency.

Table I shows that the ADCs account for 58% of tile power and 31% of tile area. No other component takes up more than 15% of tile power. Scaling the ISAAC system in the future would largely be dictated by the ADC energy scaling, which has historically been improving 2× every 2.6 years, being only slightly worse than Moore's Law scaling for digital circuits [1]. The eDRAM buffer and the eDRAM-IMA bus together take up 47% of tile area. Many of the supporting digital units (shift-and-add, MaxPool, Sigmoid, SRAM buffers) take up negligibly small amounts of area and power.

For the SE metric, we identify the optimal ISAAC-SE design in Figure 5. We note that ISAAC-CE, ISAAC-PE, and ISAAC-SE have SE values of 0.96 MB/mm^2, 1 MB/mm^2, and 54.79 MB/mm^2, respectively. It is worth noting that while ISAAC-SE cannot achieve the performance and energy of ISAAC-CE and ISAAC-PE, it can implement a large network with fewer chips. For example, the large DNN benchmark can fit in just one ISAAC-SE chip, while it needs 32 ISAAC-CE chips, and 64 DaDianNao chips. This makes ISAAC-SE an attractive design point when constrained by cost or board layouts.

As a sensitivity, we also consider the impact of moving to 32-bit fixed point computations, while keeping the same bus bandwidth. This would reduce overall throughput by 4× since latency per computation and storage requirements are both doubled. Similarly, if crossbar latency is assumed to be 200 ns, throughput is reduced by 2×, but because many of the structures become simpler, CE is only reduced by 30%.

Impact of Pipelining

ISAAC's pipeline enables a reduction in the buffering requirements between consecutive layers and an increase in throughput. The only downside is an increase in power because pipelining allows all layers to be simultaneously busy.

Table III shows the input buffering requirements for the biggest layers in our benchmarks with and without pipelining. In an ISAAC design without pipelining, we assume a central buffer that stores the outputs of the currently executing layer. The size of this central buffer is the maximum buffer requirement for any layer in Table III, viz, 1,176 KB. With the ISAAC pipeline, the buffers are scattered across the tiles. For our benchmarks, we observe that no layer requires more than 74 KB of input buffer capacity. The layers with the largest input buffers also require multiple tiles to store their synaptic weights. We are therefore able to establish 64 KB as the maximum required size for the eDRAM buffer in a tile. This small eDRAM size is an important contributor to the high CE and SE achieved by ISAAC. Similarly, based on our benchmark behaviors, we estimate that the inter-tile link bandwidth requirement never exceeds 3.2 GB/s. We therefore conservatively assume a 32-bit link operating at 1 GHz.

N_i, k ,N_x	No pipeline (KB)	With pipeline (KB)
VGG and MSRA		
3,3,224	147	1.96
96,7,112	**1176**	**74**
64,3,112	784	21
128,3,56	392	21
256,3,28	196	21
384,3,28	294	32
512,3,14	98	21
768,3,14	150	32
Deep Face		
142,11,32	142	48
71,3,32	71	6.5
63,9,16	15.75	8.8
55,9,16	13.57	7.7
25,7,16	6.25	2.7

TABLE III

BUFFERING REQUIREMENT WITH AND WITHOUT PIPELINING FOR THE LARGEST LAYERS.

The throughput advantage and power overhead of pipelining is a direct function of the number of layers in the benchmark. For example, VGG-1 has 16 layers and the pipelined version is able to achieve a throughput improvement of 16× over an unpipelined version of ISAAC. The power consumption within the tiles also increases roughly 16× in VGG-1, although, system power does not increase as much because of the constant power required by the HyperTransport links.

Impact of Data Layout, ADCs/DACs, and Noise

As the power breakdown in Table I shows, the 96 ADCs in a tile account for 58% of tile power. Recall that we had

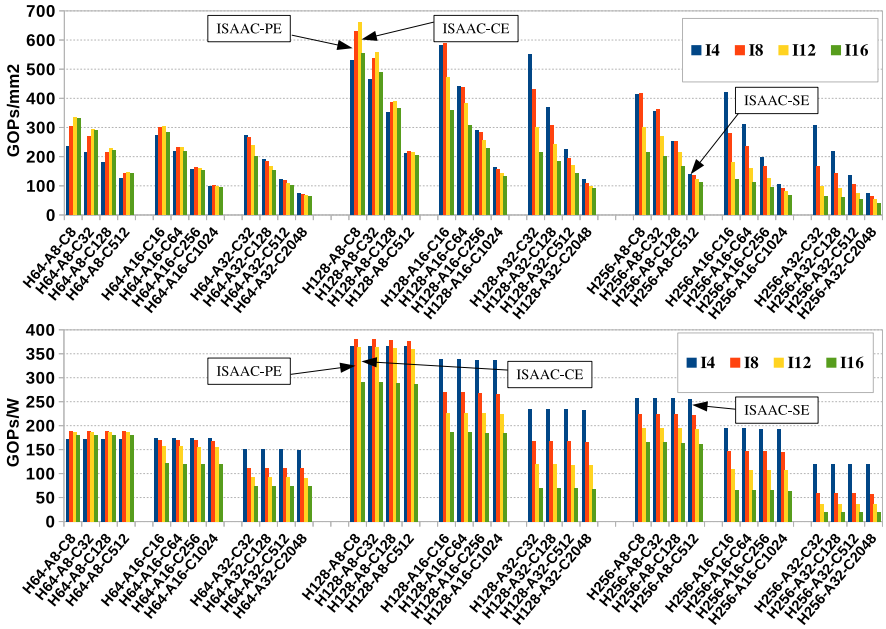


Fig. 5. CE and PE numbers for different ISAAC configurations. H128-A8-C8 for bar I12 corresponds to a tile with 12 IMAs, 8 ADCs per IMA, and 8 crossbar arrays of size 128×128 per IMA.

carefully mapped data to the memristor arrays to reduce these overheads – we now quantitatively justify those choices.

In our analysis, we first confirmed that a 9-bit ADC is never worth the power/area overhead. Once we limit ourselves to using an 8-bit ADC, a change to the DAC resolution or the bits per cell is constrained by Equations 1 and 2. If one of these (v or w) is increased linearly, the number of rows per array R must be reduced exponentially. This reduces throughput per array. But it also reduces the eDRAM/bus overhead because the IMA has to be fed with fewer inputs. We empirically estimated that the CE metric is maximized when using 2 bits per cell ($w = 2$). Performing a similar analysis for DAC resolution v yields maximal CE for 1-bit DACs. Even though v and w have a symmetric impact on R and eDRAM/bus overheads, the additional overhead of multi-bit DACs pushes the balance further in favor of using a small value for v. In particular, using a 2-bit DAC increases the area and power of a chip by 63% and 7% respectively, without impacting overall throughput. Similarly, going from a 2-bit cell to a 4-bit cell reduces CE and PE by 23% and 19% respectively.

The encoding scheme introduced in Section V allows the use of a lower resolution 8-bit ADC. Without the encoding scheme, we would either need a 9-bit ADC or half as many rows per crossbar array. The net impact is that the encoding scheme enables a 50% and 87% improvement in CE and PE, respectively. Because of the nature of Equation 2, a linear change to ADC resolution has an exponential impact on throughput; therefore, saving a single bit in our computations is very important.

Circuit parasitics and noise are key challenges in designing mixed-signal systems. Hu et al. [26] demonstrate 5 bits per cell and a 512×512 crossbar array showing no accuracy degradation compared to a software approach for the MNIST dataset, after considering thermal noise in memristor, short noise in circuits, and random telegraphic noise in the crossbar. Our analysis yields optimal throughput with a much simpler and conservative design: 1-bit DAC, 2 bits per cell, and 128×128 crossbar arrays. A marginal increase in signal noise can be endured given the inherent nature of CNNs to tolerate noisy input data.

B. Comparison to DaDianNao

Table IV compares peak CE, PE, and SE values for DaDianNao and ISAAC. DaDianNao's NFU unit has high computational efficiency – 344 $GOPS/s \times mm^2$. Adding eDRAM banks, central I/O buffers, and interconnects brings CE down to 63.5 $GOPS/s \times mm^2$. On the other hand, a 128×128 memristor array with 2 bits per cell has a CE of 1707 $GOPS/s \times mm^2$. In essence, the crossbar is a very efficient way to perform a bulk of the necessary bit arithmetic (e.g., adding 128 sets of 128 2-bit numbers) in parallel, with a few shift-and-add circuits forming the tail end of the necessary arithmetic (e.g., merging 8 numbers). Adding ADCs, other tile overheads, and the eDRAM buffer brings the CE down to 479 $GOPS/s \times mm^2$. The key to ISAAC's superiority in CE is the fact that the memristor array has very high computational parallelism, *and* storage density. Also, to perform any computation, DaDianNao fetches two values from eDRAM, while ISAAC has to only fetch one input from eDRAM (thanks to in-situ computing).

An ISAAC chip consumes more power (65.8 W) than a DaDianNao chip (20.1 W) of the same size, primarily because it has higher computational density and because of ADC overheads. In terms of PE though, ISAAC is able to beat DaDianNao by 27%. This is partially because DaDianNao

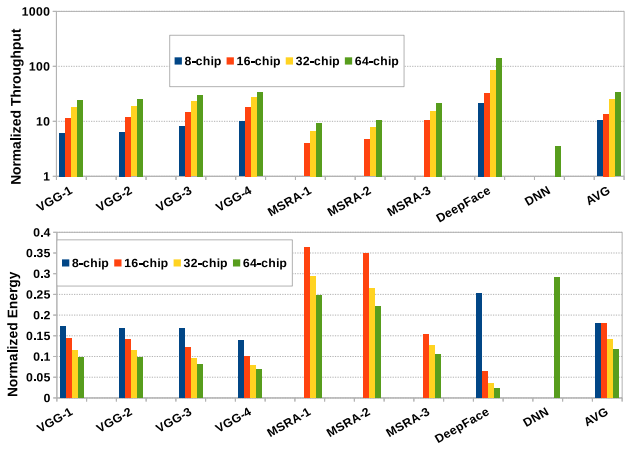


Fig. 6. Normalized throughput (top) and normalized energy (bottom) of ISAAC with respect to DaDianNao.

pays a higher "tax" on HyperTransport power than ISAAC. The HT is a constant overhead of 10 W, representing half of DaDianNao chip power, but only 16% of ISAAC chip power.

Architecture	CE $GOPs/(s \times mm^2)$	PE $GOPs/W$	SE MB/mm^2
DaDianNao	63.46	286.4	0.41
ISAAC-CE	478.95	363.7	0.74
ISAAC-PE	466.8	380.7	0.71
ISAAC-SE	140.3	255.3	54.8

TABLE IV
COMPARISON OF ISAAC AND DADIANNAO IN TERMS OF CE, PE, AND SE. HYPERTRANSPORT OVERHEAD IS INCLUDED.

Next, we execute our benchmarks on ISAAC-CE and DaDianNao, while assuming 8-, 16-, 32-, and 64-chip boards. Figure 6a shows throughputs for our benchmarks for each case. We don't show results for the cases where DaDianNao chips do not have enough eDRAM storage to accommodate all synaptic weights. In all cases, the computation spreads itself across all available IMAs (ISAAC) or NFUs (DaDianNao) to maximize throughput. Figure 6b performs a similar comparison in terms of energy.

For large benchmarks on many-chip configurations, DaDianNao suffers from the all-to-all communication bottleneck during the last classifier layers. As a result, it has low NFU utilization in these layers. Meanwhile, ISAAC has to replicate the weights in early layers several times to construct a balanced pipeline. In fact, in some benchmarks, the first layer has to be replicated more than 50K times to keep the last layer busy in every cycle. Since we don't have enough storage for such high degrees of replication, the last classifier layers also see relatively low utilization in the IMAs. Thus, on early layers, ISAAC has a far higher CE than early layers of DaDianNao – the actual speedups vary depending on the degree of replication for each layer. In later layers, both operate at low utilization – DaDianNao because of bandwidth limitations and ISAAC because of limitations on replication. The net effect of these phenomena is that on average for 16-chip configurations, ISAAC-CE achieves 14.8× higher throughput, consumes 95% more power, and achieves 5.5× lower energy than DaDianNao.

While we don't compare against state-of-the-art GPU implementations of CNNs, we note that a 64-chip DaDianNao [9] has already been shown to have 450× speedup and 150× lower energy than an NVIDIA K20M GPU.

IX. RELATED WORK

Accelerators. Accelerators have been designed for several application domains, e.g., databases [76], Memcached [40], and machine learning algorithms for image [34], [38] and speech [19] recognition. DianNao [8] highlights the memory wall and designs an accelerator for CNNs and DNNs that exploit data reuse with tiling. The inherent sharing of weights in CNNs is explored in ShiDianNao [13]. Origami [7] is an ASIC architecture that tries to reduce I/O Bandwidth for CNNs. The PuDianNao accelerator [41] focuses on a range of popular machine learning algorithms and constructs common computational primitives. CNNs have also been mapped to FPGAs [11], [15], [16]. The Convolution Engine [55] identifies data flow and locality patterns that are common among kernels of image/video processing and creates custom units that can be programmed for different applications. Minerva [57] automates a co-design approach across algorithm, architecture, and circuit levels to optimize digital DNN hardware. RedEye [39] moves processing of convolution layers to an image sensor's analog domain to reduce computational burden. Neurocube [31] maps CNNs to 3D high-density high-bandwidth memory integrated with logic, forming a mesh of digital processing elements. The sparsity in CNN parameters and activations are leveraged by EIE [62] to design an accelerator for sparse vector-matrix multiplication.

Neuromorphic Computing. A related body of neural-inspired architectures includes IBM's neurosynaptic core [44] that emulates biological neurons. At the moment, these architectures [27], [44], [47], based on spiking neural networks, are competitive with CNNs and DNNs in terms of accuracy in a limited number of domains.

Memristors. Memristors [66] have been primarily targeted as main memory devices [25], [74], [77]. Memristor crossbars have been recently proposed for analog dot product computations [32], [51]. Studies using SPICE models [71] show that an analog crossbar can yield higher throughput and lower power than a traditional HPC system. Mixed-signal computation capabilities of memristors have been used to speed up neural networks [33], [42], [43], [53]. In-situ memristor computation has also been used for perceptron networks to recognize patterns in small scale images [78]. Bojnordi et al. design an accelerator for Boltzmann machine [4], using in-situ memristor computation to estimate a sum of products of two 1-bit numbers and a 16-bit weight. A few works have attempted training on memristors [2], [53], [65]. Other emerging memory technologies (e.g., PCM) have also been used as synaptic weight elements [6], [68]. None of the above works are targeted at CNNs or DNNs. Chi et al. [10] propose PRIME, a morphable PIM structure for ReRAM based main memory with carefully designed peripheral circuits that allow arrays to be used as memory, scratchpads, and dot product engines for CNN workloads. The data encoding and pipelining in ISAAC and PRIME are both very different. While PRIME can support positive and negative synapses, input vectors must be unsigned. Also, the dot-product computations in PRIME are

lossy because the precision of the ADC does not always match the precision of the computed dot-product.

Hardware Neural Networks. A number of works have designed hardware neural networks and neuromorphic circuits using analog computation [5], [29], [52], [60], [61], [72] to take advantage of faster vector-matrix multiplication [18], [56]. Neural network accelerators have also been built for signal processing [3] with configurable digital-analog models, or for defect tolerant cores [14], [22], [73]. Neural models have been used to speedup approximate code [20] using limited precision analog hardware [64].

Large Scale Computing. DjiNN [23] implements a high throughput DNN service cloud infrastructure using GPU server designs. Microsoft has published a whitepaper [50] on extending the Catapult project [54] to deploy CNN workloads on servers augmented with FPGAs.

X. Conclusions

While the potential for crossbars as analog dot product engines is well known, our work has shown that a number of challenges must be overcome to realize a full-fledged CNN architecture. In particular, a balanced inter-layer pipeline with replication, an intra-tile pipeline, efficient handling of signed arithmetic, and bit encoding schemes are required to deliver high throughput and manage the high overheads of ADCs, DACs, and eDRAMs. We note that relative to DaDianNao, ISAAC is able to deliver higher peak computational and power efficiency because of the nature of the crossbar, and in spite of the ADCs accounting for nearly half the chip power. On benchmark CNNs and DNNs, we observe that ISAAC is able to out-perform DaDianNao significantly in early layers, while the last layers suffer from under-utilization in both architectures. On average for a 16-chip configuration, ISAAC is able to yield a 14.8× higher throughput than DaDianNao.

Acknowledgment

This work was supported in parts by Hewlett Packard Labs, US Department of Energy (DOE) under Cooperative Agreement no. DE-SC0012199, University of Utah, and NSF grants 1302663 and 1423583. In addition, John Paul Strachan, Miao Hu, and R. Stanley Williams acknowledge support in part from the Intelligence Advanced Research Projects Activity (IARPA) via contract number 2014-14080800008.

References

[1] "ADC Performance Evolution: Walden Figure-Of-Merit (FOM)," 2012, https://converterpassion.wordpress.com/2012/08/21/adc-performance-evolution-walden-figure-of-merit-fom/.

[2] F. Alibart, E. Zamanidoost, and D. B. Strukov, "Pattern Classification by Memristive Crossbar Circuits using Ex-Situ and In-Situ Training," *Nature*, 2013.

[3] B. Belhadj, A. Joubert, Z. Li, R. Héliot, and O. Temam, "Continuous Real-World Inputs Can Open Up Alternative Accelerator Designs," in *Proceedings of ISCA-40*, 2013.

[4] M. N. Bojnordi and E. Ipek, "Memristive Boltzmann Machine: A Hardware Accelerator for Combinatorial Optimization and Deep Learning," in *Proceedings of HPCA-22*, 2016.

[5] B. E. Boser, E. Sackinger, J. Bromley, Y. Le Cun, and L. D. Jackel, "An Analog Neural Network Processor with Programmable Topology," *Journal of Solid-State Circuits*, 1991.

[6] G. Burr, R. Shelby, C. di Nolfo, J. Jang, R. Shenoy, P. Narayanan, K. Virwani, E. Giacometti, B. Kurdi, and H. Hwang, "Experimental Demonstration and Tolerancing of a Large-Scale Neural Network (165,000 Synapses), using Phase-Change Memory as the Synaptic Weight Element," in *Proceedings of IEDM*, 2014.

[7] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A Convolutional Network Accelerator," in *Proceedings of GLSVLSI-25*, 2015.

[8] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in *Proceedings of ASPLOS*, 2014.

[9] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of MICRO-47*, 2014.

[10] P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *Proceedings of ISCA-43*, 2016.

[11] J. Cloutier, S. Pigeon, F. R. Boyer, E. Cosatto, and P. Y. Simard, "VIP: An FPGA-Based Processor for Image Processing and Neural Networks," 1996.

[12] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep Learning with COTS HPC Systems," in *Proceedings of ICML-30*, 2013.

[13] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *Proceedings of ISCA-42*, 2015.

[14] Z. Du, A. Lingamneni, Y. Chen, K. Palem, O. Temam, and C. Wu, "Leveraging the Error Resilience of Machine-Learning Applications for Designing Highly Energy Efficient Accelerators," in *Proceedings of ASPDAC-19*, 2014.

[15] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A Runtime Reconfigurable Dataflow Processor for Vision," in *Proceedings of CVPRW*, 2011.

[16] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based Processor for Convolutional Networks," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2009.

[17] J. Fieres, K. Meier, and J. Schemmel, "A Convolutional Neural Network Tolerant of Synaptic Faults for Low-Power Analog Hardware," in *Proceedings of Artificial Neural Networks in Pattern Recognition*, 2006.

[18] R. Genov and G. Cauwenberghs, "Charge-Mode Parallel Architecture for Vector-Matrix Multiplication," 2001.

[19] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *Proceedings of ICASSP*, 2013.

[20] B. Grigorian, N. Farahpour, and G. Reinman, "BRAINIAC: Bringing Reliable Accuracy Into Neurally-Implemented Approximate Computing," in *Proceedings of HPCA-21*, 2015.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," *arXiv preprint arXiv:1502.02551*, 2015.

[22] A. Hashmi, H. Berry, O. Temam, and M. Lipasti, "Automatic Abstraction and Fault Tolerance in Cortical Microachitectures," in *Proceedings of ISCA-38*, 2011.

[23] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "DjiNN and Tonic: DNN as a Service and Its Implications for Future Warehouse Scale Computers," in *Proceedings of ISCA-42*, 2015.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv preprint arXiv:1502.01852*, 2015.

[25] Y. Ho, G. M. Huang, and P. Li, "Nonvolatile Memristor Memory: Device Characteristics and Design Implications," in *Proceedings of ICCAD-28*, 2009.

[26] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, R. S. Williams, and J. Yang, "Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication," in *Proceedings of DAC-53*, 2016.

[27] T. Iakymchuk, A. Rosado-Muñoz, J. F. Guerrero-Martínez, M. Bataller-Mompeán, and J. V. Francés-Víllora, "Simplified Spiking Neural Network Architecture and STDP Learning Algorithm Applied to Image Classification," *Journal on Image and Video Processing (EURASIP)*, 2015.

[28] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the Best Multi-Stage Architecture for Object Recognition?" in *Proceedings of ICCV-12*, 2009.

[29] A. Joubert, B. Belhadj, O. Temam, and R. Héliot, "Hardware Spiking Neurons Design: Analog or Digital?" in *Proceedings of IJCNN*, 2012.

[30] O. Kavehei, S. Al-Sarawi, K.-R. Cho, N. Iannella, S.-J. Kim, K. Eshraghian, and D. Abbott, "Memristor-based Synaptic Networks and Logical Operations Using In-Situ Computing," in *Proceedings of ISSNIP*, 2011.

[31] D. Kim, J. H. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *Proceedings of ISCA-43*, 2016.

[32] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A Functional Hybrid Memristor Crossbar-Array/CMOS System for Data Storage and Neuromorphic Applications," *Nano Letters*, 2011.

[33] Y. Kim, Y. Zhang, and P. Li, "A Digital Neuromorphic VLSI Architecture with Memristor Crossbar Synaptic Array for Machine Learning," in *Proceedings of SOCC-3*, 2012.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of NIPS*, 2012.

[35] C. Kügeler, C. Nauenheim, M. Meier, R. Waser *et al.*, "Fast Resistance Switching of TiO 2 and MSQ Thin Films for Non-Volatile Memory Applications (RRAM)," in *Proceedings of NVMTS-9*, 2008.

[36] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Brandli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1 mW 8b 1.2 GS/s Single-Channel Asynchronous SAR ADC with Alternate Comparators for Enhanced Speed in 32 nm Digital SOI CMOS," *Journal of Solid-State Circuits*, 2013.

[37] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng, "Building High-Level Features using Large Scale Unsupervised Learning," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[38] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, 1998.

[39] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision," in *Proceedings of ISCA-43*, 2016.

[40] K. Lim, D. Meisner, A. Saidi, P. Ranganathan, and T. Wenisch, "Thin Servers with Smart Pipes: Designing Accelerators for Memcached," in *Proceedings of ISCA*, 2013.

[41] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "PuDianNao: A Polyvalent Machine Learning Accelerator," in *Proceedings of ASPLOS-20*.

[42] X. Liu, M. Mao, H. Li, Y. Chen, H. Jiang, J. J. Yang, Q. Wu, and M. Barnell, "A Heterogeneous Computing System with Memristor-based Neuromorphic Accelerators," in *Proceedings of HPEC-18*, 2014.

[43] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu *et al.*, "RENO: A High-Efficient Reconfigurable Neuromorphic Computing Accelerator Design," in *Proceedings of DAC-52*, 2015.

[44] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha, "A Digital Neurosynaptic Core Using Embedded Crossbar Memory with 45pJ per Spike in 45nm," in *Proceedings of CICC*, 2011.

[45] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *Proceedings of MICRO*, 2007.

[46] B. Murmann, "ADC Performance Survey 1997-2015 (ISSCC & VLSI Symposium)," 2015, http://web.stanford.edu/∼murmann/adcsurvey.html.

[47] A. Nere, A. Hashmi, M. Lipasti, and G. Tononi, "Bridging the Semantic Gap: Emulating Biological Neuronal Behaviors with Simple Digital Neurons," in *Proceedings of HPCA-19*, 2013.

[48] M. O'Halloran and R. Sarpeshkar, "A 10-nW 12-bit Accurate Analog Storage Cell with 10-aA Leakage," *Journal of Solid-State Circuits*, 2004.

[49] W. Ouyang, P. Luo, X. Zeng, S. Qiu, Y. Tian, H. Li, S. Yang, Z. Wang, Y. Xiong, C. Qian *et al.*, "DeepId-Net: Multi-Stage and Deformable Deep Convolutional Neural Networks for Object Detection," *arXiv preprint arXiv:1409.3505*, 2014.

[50] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating Deep Convolutional Neural Networks Using Specialized Hardware," 2015, http://research.microsoft.com/apps/pubs/default.aspx?id=240715.

[51] Y. V. Pershin and M. Di Ventra, "Experimental Demonstration of Mssociative Memory with Memristive Neural Networks," *Neural Networks*, 2010.

[52] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "NeuFlow: Dataflow Vision Processing System-On-a-Chip," in *Proceedings of the MWSCAS-55*, 2012.

[53] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and Operation of an Integrated Neuromorphic Network based on Metal-Oxide Memristors," *Nature*, 2015.

[54] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray *et al.*, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *Proceedings of ISCA-41*, 2014.

[55] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing," in *Proceedings of ISCA-40*, 2013.

[56] S. Ramakrishnan and J. Hasler, "Vector-Matrix Multiply and Winner-Take-All as an Analog Classifier," 2014.

[57] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. Lee, J. M. Hernandez, Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling Low-Power, High-Accuracy Deep Neural Network Accelerators," in *Proceedings of ISCA-43*, 2016.

[58] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, 2014.

[59] M. Saberi, R. Lotfi, K. Mafinezhad, W. Serdijn *et al.*, "Analysis of Power Consumption and Linearity in Capacitive Digital-to-Analog Converters used in Successive Approximation ADCs," 2011.

[60] E. Sackinger, B. E. Boser, J. Bromley, Y. LeCun, and L. D. Jackel, "Application of the ANNA Neural Network Chip to High-Speed Character Recognition," *IEEE Transactions on Neural Networks*, 1991.

[61] J. Schemmel, J. Fieres, and K. Meier, "Wafer-Scale Integration of Analog Neural Networks," in *Proceedings of IJCNN*, 2008.

[62] S.Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of ISCA*, 2016.

[63] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[64] R. St Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-Purpose Code Acceleration with Limited-Precision Analog Computation," in *Proceeding of ISCA-41*, 2014.

[65] J. Starzyk and Basawaraj, "Memristor Crossbar Architecture for Synchronous Neural Networks," *Transactions on Circuits and Systems I*, 2014.

[66] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. Williams, "The Missing Memristor Found," *Nature*, vol. 453, pp. 80–83, May 2008.

[67] Y. Sun, X. Wang, and X. Tang, "Deep Learning Face Representation from Predicting 10,000 Classes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[68] M. Suri, V. Sousa, L. Perniola, D. Vuillaume, and B. DeSalvo, "Phase Change Memory for Synaptic Plasticity Application in Neuromorphic Systems," in *Proceedings of IJCNN*, 2011.

[69] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[70] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2010.

[71] T. Taha, R. Hasan, C. Yakopcic, and M. McLean, "Exploring the Design Space of Specialized Multicore Neural Processors," in *Proceedings of IJCNN*, 2013.

[72] S. M. Tam, B. Gupta, H. Castro, M. Holler *et al.*, "Learning on an Analog VLSI Neural Network Chip," in *Proceedings of the International Conference on Systems, Man and Cybernetics*, 1990.

[73] O. Temam, "A Defect-Tolerant Accelerator for Emerging High-Performance Applications," in *Proceedings of ISCA-39*, 2012.

[74] P. O. Vontobel, W. Robinett, P. J. Kuekes, D. R. Stewart, J. Straznicky, and R. S. Williams, "Writing to and reading from a nano-scale crossbar memory based on memristors," *Nanotechnology*, vol. 20, 2009.

[75] L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[76] L. Wu, R. Barker, M. Kim, and K. Ross, "Navigating Big Data with High-Throughput Energy-Efficient Data Partitioning," in *Proceedings of ISCA-40*, 2013.

[77] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the Challenges of Crossbar Resistive Memory Architectures," in *Proceedings of HPCA-21*, 2015.

[78] C. Yakopcic and T. M. Taha, "Energy Efficient Perceptron Pattern Recognition using Segmented Memristor Crossbar Arrays," in *Proceedings of IJCNN*, 2013.

[79] M. Zangeneh and A. Joshi, "Design and Optimization of Nonvolatile Multibit 1T1R Resistive RAM," *Proceedings of the Transactions on VLSI Systems*, 2014.

[80] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Proceedings of ECCV*, 2014.