HEIF: Highly Efficient Stochastic Computing based Inference Framework for Deep Neural Networks

Zhe Li, *Student Member, IEEE,* Ji Li, *Student Member, IEEE,* Ao Ren, *Student Member, IEEE,* Ruizhe Cai, *Student Member, IEEE,* Caiwen Ding, *Student Member, IEEE,* Xuehai Qian, *Member, IEEE,* Jeffrey Draper, *Member, IEEE,* Bo Yuan, *Member, IEEE,* Jian Tang, *Member, IEEE,* Qinru Qiu, *Member, IEEE,* and Yanzhi Wang, *Member, IEEE*

***Abstract*—Deep Convolutional Neural Networks (DCNNs) are one of the most promising types of deep learning technique and have been recognized as the dominant approach for almost all recognition and detection tasks. The computation of DCNNs is highly computational and memory intensive for the large feature maps and neuron connections, and the performance highly depends on the capability of hardware resources. With the recent trend of wearable devices and Internet of Things (IoTs), it becomes attractive to integrate the DCNNs onto embedded and portable devices, which require low power & energy consumptions and small hardware footprints.**

Recent work SC-DCNN [1] demonstrates that Stochastic Computing (SC), as a low-cost substitute to binary-based computing, can radically simplify the hardware implementation of arithmetic units and has the potential to satisfy the stringent power requirements in embedded devices. In SC, many arithmetic operations that are resource-consuming in binary designs can be implemented with very simple hardware logic, alleviating the extensive computational complexity. It offers a colossal design space for integration and optimization due to its reduced area and soft error resiliency.

In this paper, we present *HEIF*, a highly efficient SC-based inference framework of the large-scale DCNNs, with broad applications including (but not limited to) *LeNet-5 and AlexNet*, that achieves high energy efficiency and low area/hardware cost. Compared to SC-DCNN [1], HEIF features with *1)* the first (to the best of our knowledge) SC-based Rectified Linear Unit (ReLU) activation function to catch up with the recent advances in software models and mitigate degradation in application-level accuracy; *2)* the redesigned Approximate Parallel Counter (APC) and optimized stochastic multiplication using transmission gates and inverse mirror adders; and *3)* the new optimization of weight storage using clustering. Most importantly, to achieve maximum energy efficiency while maintaining acceptable accuracy, HEIF considers holistic optimizations on cascade connection of function blocks in DCNN, pipelining technique, and bit-stream length reduction. Experimental results show that in large-scale applications HEIF outperforms previous SC-DCNN by the throughput of $4.1\times$, by area efficiency of up to $6.5\times$ and achieves up to $5.6\times$ energy improvement.

***Keywords*—Stochastic Computing, Deep learning, Convolutional Neural Network, Energy-efficient, ASIC, Optimization.**

I. Introduction

Machine learning technology benefits many aspects of modern life: web searches, e-commerce recommendations, social network content filtering, *etc.* [2]. Unfortunately, the conventional machine learning techniques were restricted by the lack of ability to automatically extract high-level features which have been conducted by well-engineered manual feature extractors. *Deep learning* methods have taken advantage of the architecture of multi-level representations to learn very complex functions [2]. Here, each representation is obtained through the transformation from a slightly less abstract level by a simple non-linear module. Deep learning significantly enhances the machine learning capability by learning from data by these multiple layers for different features without human involvement.

Deep Convolutional Neural Networks (DCNNs) is one of the most promising types of artificial neural networks based on deep learning and have been recognized as the dominant approach for almost all recognition and detection tasks. DCNNs feature the special structural designs [3] of layer-wise local connections implementing convolution, integrating pattern matching techniques into neural networks and learning invariant elementary features of images. It has been demonstrated that DCNNs are effective models for understanding image content [4], image classification [5], video classification [4] and object detection [6,7].

Due to the deep structure, the performance of DCNN highly relies on the capability of hardware resources. From high performance server clusters [8,9] to *General-Purpose Graphics Processing Units (GPGPUs)* [10,11], parallel accelerations of DCNNs are widely used in both the academic and industry. Recently, hardware acceleration for DCNNs has attracted enormous research interests on *Field-Programmable Gate Arrays (FPGAs)* [12–14]. Nevertheless, there is a trend of embedding DCNNs into light-weight embedded and portable systems, such as surveillance monitoring systems [15], self-driving systems [16], unmanned aerial systems [17], and robotic systems [18]. These scenarios require very low power & energy consumptions and small hardware footprints. Besides, cell phones [2] and wearable devices [19] equipped with hardware-level neural network computation capability require the radical reduction in power & energy consumptions and footprints.

DCNNs are both compute and memory intensive. Based on the conventional binary arithmetic calculations (used in prior GPU, FPGA and ASIC accelerators), deploying the entire large DCNNs like AlexNet [20–22] (for ImageNet applications) incurs the significant amount of hardware, power and energy cost. This makes it impractical to use DCNNs in embedded systems with a limited area and power budget. Therefore, the novel alternative computing paradigms are urgently needed to overcome this hurdle.

The recent work [1] considered *Stochastic Computing (SC)*,

Zhe Li, Ao Ren, Ruizhe Cai, Caiwen Ding, Jian Tang, Qinru Qiu, and Yanzhi Wang are with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, 13244.
(E-mail: {zli89, aren, rcai100, cading, jtang02, qiqiu, ywang393}@syr.edu)

Ji Li, Xuehai Qian, and Jeffrey Draper are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, 90089, USA (E-mail: jli724, xuehai.qian@usc.edu, and draper@isi.edu).

Bo Yuan is with the the Department of Electrical Engineering at City University of New York (CUNY), City College (CCNY), New York, NY 10031, USA (E-mail: byuan@ccny.cuny.edu.

a special approximate computing technique such as [23–28], as a low-cost substitute to binary-based computing [29] for DCNNs. SC can radically simplify the hardware implementation of arithmetic units and has the potential to satisfy the low-power requirements of DCNNs. In SC, many arithmetic operations that are resource-consuming in binary designs can be implemented with very simple hardware logic [30], alleviating the extensive computation complexity. It offers a colossal design space for optimization due to its reduced area and soft error resiliency. Recent works [31–33] applied SC to neural networks and *Deep Belief Networks (DBNs)*, demonstrating the applicability of SC on deep learning techniques.

Unlike DBNs, implementing DCNNs using SC is more challenging due to local connectivities, down-sampling operations and special activation functions, i.e., the *Rectified Linear Unit (ReLU)* function [3,4]. SC-DCNN [1] is the first to investigate SC-based DCNN design space explorations. It does have the following limitations. First, SC-DCNN suffers from the degraded overall accuracy because it utilizes the easy-to-implement hyperbolic tangent (tanh) function instead of ReLU function. Second, SC-DCNN is not sufficiently optimized, which leads to *1)* the difficulty to maintain the high application-level accuracy due to the stochastic nature of SC components; and most importantly, *2)* a low clock frequency of *no more than 200MHz*. To overcome these limitations and further improve energy efficiency, we present *HEIF* (i.e. **H**ighly **E**fficient **I**nference **F**ramework) with broad applications including (but not limited to) *LeNet5 and AlexNet*, that achieves high energy efficiency and low area/hardware cost. HEIF includes the following *key innovations*:

1) We propose the *first* (to the best of our knowledge) SC-based ReLU activation function and corresponding optimizations to catch up with recent software advances and mitigate degradation on application-level accuracy.

2) We re-design the *Approximate Parallel Counter (APC)* proposed in [34] and optimize stochastic multiplication, which is utilized in the inner product calculations of DCNN, to achieve a smaller footprint and higher energy efficiency without sacrificing any precision.

3) We investigate a memory reduction and clustering method considering the effects of hardware imprecision on the overall application-level accuracy.

4) HEIF is holistically optimized with the cascade structural connection of function blocks, the pipelining technique, and the bit-stream length reduction. It significantly improves the energy efficiency without compromising application-level accuracy requirements.

Overall, HEIF could achieve very high energy efficiency of $\underline{1.2M}$ Images/J and $\underline{1.3M}$ Images/J, and high throughput of $\underline{3.2M}$ Images/s and $\underline{2.5M}$ Images/s, along with very small area of $\underline{22.9}$ mm^2 and $\underline{24.7}$ mm^2 on LeNet-5 and AlexNet, respectively. HEIF outperforms SC-DCNN [1] by throughput of $\underline{4.1\times}$, by area efficiency of up to $\underline{6.5\times}$ and achieves up to $\underline{5.6\times}$ energy improvement.

II. Preliminary Work

A. DCNN Architecture Overview

Deep Convolutional Neural Networks (DCNN) are biologically inspired variants of *multi-layer perceptrons (MLPs)* by mimicking the animal visual mechanism [35]. Thus, a DCNN has special sets of neurons only connected to a small receptive field of its previous layer rather than fully connected. Besides

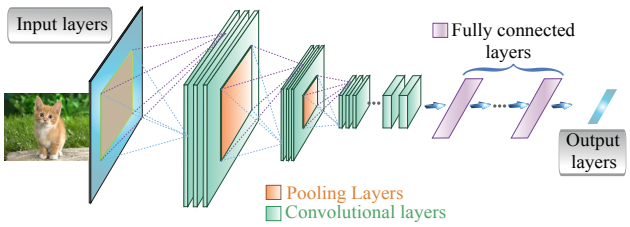


Fig. 1. **General DCNN architecture.**

an input layer and an output layer, a general DCNN architecture consists of a stack of *convolutional layers*, *pooling layers*, and *fully connected layers* shown in Figure 1. Please note that some special layers like normalization or regularization are not the focus in this paper.

1) A convolutional layer is associated with a set of learnable filters (or kernels) [3], which are activated when specific types of features are found at some spatial positions in the inputs. Filter-sized moving windows are applied to the inputs to obtain a set of feature maps by calculating the convolution of the filter and inputs in the moving window. Each *convolutional neuron*, representing one pixel in a feature map, takes a set of inputs and corresponding filter weights to calculate their inner-products.

2) After extracting features using convolution, a subsampling step can be applied to aggregate statistics of these features to reduce the dimensions of data and mitigate overfitting issues. This subsampling operation is realized by a *pooling neuron* in pooling layers, where different non-linear functions can be applied, such as max pooling, average pooling, and L2-norm pooling. Among them, max pooling is the dominating type of pooling in state-of-the-art DCNNs due to the higher overall accuracy and convergence speed. The activation functions are non-linear transformation functions, such as Rectified Linear Units (ReLU) $f(x) = max(0, x)$, hyperbolic tangent (tanh) $f(x) = tanh(x)$ or $f(x) = |tanh(x)|$, and sigmoid function $f(x) = \frac{1}{1+e^{-x}}$. Among them, the ReLU function is the dominating type in the (large-scale) DCNNs due to *i)* the lower complexity for software implementation; and *ii)* the reduced vanishing gradient problem [36]. These non-linear transformations are conducted somewhere before the inputs of the next layer, ensuring that they are within the range of $[-1, 1]$. Usually, a combination of convolutional neurons, pooling neurons and activation functions forms a *feature extraction block (FEB)* to extract high-level abstraction from the input images or previous low-level features.

3) A fully connected layer is a normal neural network layer with its inputs fully connected with its previous layer. Each *fully connected neuron* calculates the inner-product of its inputs and corresponding weights.

In general, a DCNN inference process has three basic *function blocks* shown in Figure 2. *1)* The *inner-product* (Figure 2 (a)) of inputs and weights corresponding to their incoming connections with the previous layer is calculated by neurons in convolutional layers and fully connected layers; *2)* The *pooling* block (Figure 2 (b)) sub-samples the inner-products; *3)* The *activation function* block (Figure 2 (c)) transforms the inner-products or sub-sampled outputs to ensure that the inputs of next layer are within the valid range.

The overall application-level accuracy (e.g., the overall classification rates) is one of the key optimization goals of the SC-based DCNN. On the other hand, the SC-based function

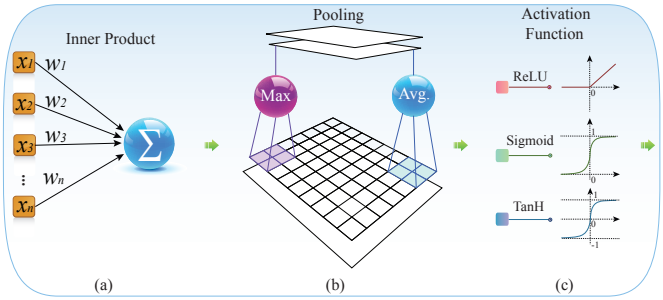


Fig. 2. **Function blocks in a DCNN. (a) Inner-product, (b) pooling, and (c) activation.**

blocks and FEBs exhibit a certain degree of imprecision due to the inherent stochastic nature. The application-level accuracy and hardware precision are different but correlated, which implies the high precision in each function block will likely lead to a high overall application-level accuracy. Therefore, the hardware precisions will be optimized for the SC-based function blocks and FEBs.

B. Stochastic Computing

In SC, a probabilistic number x in the range of [0,1] is represented by a sequence of binary digits X (i.e, a bit-stream), where the value of x is contained in the primary statistic of the bit-stream or the probability of any given bit in the sequence being a logic one [31]. For instance, the value of a 5-bit sequence $X = 10110$ is $x = P_{X=1} = \frac{3}{5} = 0.6$. In addition to this unipolar encoding format, SC has the bipolar encoding format to represent a number x in the range of [-1, 1], where $x = 2 \cdot P_{X=1} - 1$. For example, a sequence $X = 11101$ represents $x = 0.6$ in the bipolar format. We adopt the bipolar encoding format since the numbers in a typical DCNN are distributed on both sides of zero.

SC has three characteristics. First, only a *subset* of the real numbers can be represented exactly in SC, i.e., an m-bit sequence can only represent $\{\frac{0}{m}, \frac{1}{m}, ..., \frac{m}{m}\}$ in the unipolar format. Therefore, increasing the length of the bit-stream can improve the precision. Since the bits in the bit-stream are independent of each other, the precision can be adjusted without hardware modification, which is known as the progressive precision characteristic [29]. Second, the representation of a stochastic number is *not unique*, e.g., there are $C_5^3 = 10$ possible ways to represent 0.6 using a 5-bit SC sequence. Third, as the weight of each bit in the bit-stream is even, SC is *naturally resilient to soft errors*.

The basic arithmetic operations in DCNNs are multiplication, addition, and nonlinear activation, which can be implemented efficiently using SC with small circuits and significantly improved energy & power efficiency.

Multiplication. Stochastic multiplication can be performed efficiently by an AND gate and an XNOR gate in unipolar and bipolar format, respectively. Figure 3 (a) and (b) give the example for unipolar and bipolar multiplication. We assume that the inputs are independent of each other. For unipolar multiplication $x = P_{X=1} = P_{A_1=1} \cdot P_{A_2=1} = a_1 \cdot a_2$, whereas for bipolar multiplication $x = 2P_{X=1} - 1 = 2(P_{A_1=1} \cdot P_{A_2=1} + P_{A_1=0} \cdot P_{A_2=0}) - 1 = 2[P_{A_1=1} \cdot P_{A_2=1} + (1 - P_{A_1=1}) \cdot (1 - P_{A_2=1})] - 1 = (2P_{A_1=1} - 1) \cdot (2P_{A_2=1} - 1) = a_1 \cdot a_2$. Clearly, multiplication in SC consumes much less hardware and offers significantly improved energy & power efficiency, compared with conventional binary arithmetic.

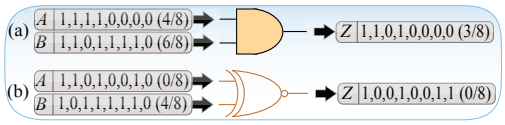


Fig. 3. **Stochastic multiplication. (a) Unipolar and (b) bipolar.**

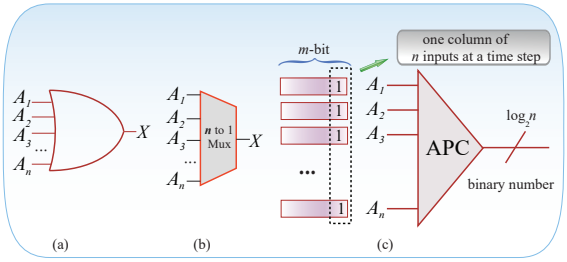


Fig. 4. **Stochastic addition using: (a) OR gate, (b) MUX, and (c) APC.**

Addition. In the SC domain, addition can be implemented by an OR gate, a *multiplexer (MUX)*, and *an Approximate Parallel Counter (APC)* [34], as shown in Figure 4 (a), (b), (c), respectively. OR gate based addition is an approximation of unipolar addition, i.e., $x = P_X \approx P_{A_1} + P_{A_2} + \cdots P_{A_n} \approx a_1 + a_2 + \cdots + a_n$, which is not suitable for the bipolar encoding format in this work. MUX-based adder works for both unipolar and bipolar formats, where a MUX is used to randomly select one input i among n inputs with probability p_i such that $\Sigma_{i=1}^{n} p_i = 1$. For example, adding two numbers using MUX is $x = 2 \cdot P_X - 1 = \frac{1}{2} \cdot \big((2 \cdot P_{A_1} - 1) + (2 \cdot P_{A_2} - 1)\big) = \frac{1}{2} \cdot (a_1 + a_2)$ in bipolar format. Since only one bit is utilized at a time, MUX-based adder has low precision when input number n is large, making it less attractive for the large DCNNs.

As shown in Figure 4 (c), [34] proposed the APC design with high precision and no bias, which calculates the summation of multiple input bit streams $(A_1 - A_n)$ by accumulating the number of 1's at each time step. Unlike the MUX-based adder, which incurs significant accuracy loss since the 1-bit wide output can only represent a number in the range of [-1,1], the output of the APC is a $log_2(n)$-bit wide binary bit-stream, which is capable of representing numbers in a wide range. As state-of-the-art DCNNs include the large filters and huge connections with the fully-connected layers (i.e., a large number of input bit streams for an adder), it becomes imperative to use APC-based addition in practice instead of MUX or OR gates. The APC should be further optimized to achieve a smaller footprint and higher energy efficiency without sacrificing precision.

Activation. Nonlinear activation function not only affects the learning dynamics but also has a significant impact on the network's expressive power [37]. Traditional activation functions, such as sigmoid $(f(x) = \frac{1}{1+e^{-x}})$ and hyperbolic tangent $(f(x) = \frac{2}{1+e^{-2 \cdot x}} - 1)$, suffer from the vanishing gradient problem, resulting in a slower training process or a convergence to a poor local minimum [38]. On the other hand, ReLU function $(f(x) = max(0, x))$ has two major benefits: *i)* the reduced likelihood of the gradient to vanish, since an activated unit gives a constant gradient of 1, and *ii)* the induced high sparsity in the hidden layers as $x \leq 0$ leads to $f(x) = 0$.

Nevertheless, to the best of our knowledge, only two types of hyperbolic tangent activation function have been designed in the SC domain for neural networks [31,32]. As shown in Figure 5 (a), Stanh(·) is designed in [31] for input bit-stream X

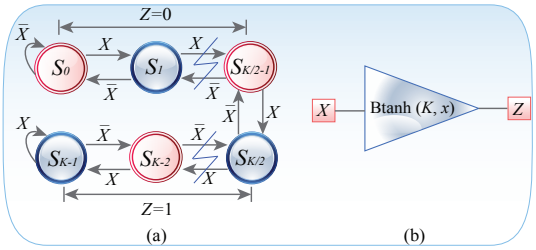


Fig. 5. **Stochastic hyperbolic tangent. (a) Stanh(·) and (b) Btanh(·).**

using a *finite state machine (FSM)* with K states. The output stream Z is determined by the current state s_i $(0 \leq i \leq K-1)$, which is calculated as

$$s_i = \begin{cases} 0, & \text{if } 0 \leq i \leq \frac{K}{2} - 1 \\ 1, & \text{otherwise} \end{cases} \quad (1)$$

The detailed mathematical explanation of $Stanh(\cdot)$ is given in [39]. On the other hand, Btanh(·) is proposed in [32] for n-input binary bit-streams with m-bit length using a two-state up/down counter, as shown in Figure 5 (b).

As ReLU has become the most popular activation function for most recent DCNNs like AlexNet [2], it is imperative to design novel SC-based ReLU activation for the state-of-the-art DCNNs. We need to resolve two challenges in developing effective SC-based ReLU: *i)* realize the nonlinear shape of ReLU using SC, and *ii)* achieve sufficient precision level. The latter is particularly important because an inaccurate activation can potentially amplify the imprecision of features after pooling.

III. Proposed Design

A. Motivation

The hundreds of millions of connections and millions of neurons in the state-of-the-art DCNNs make DCNNs both highly computational and memory intensive. In order to deploy DCNNs onto mobile systems, wearable devices, and unmanned systems, further energy efficiency enhancements must be achieved to implement the large state-of-the-art DCNN such as AlexNet [5], which is composed of over 0.65 million neurons with varying shapes across eight layers.

In the traditional binary arithmetic calculation blocks used in most of the prior GPU, FPGA, and ASIC accelerator works, the most intensive calculations in DCNNs are related to the inner-product operation in both convolution and fully connected layers. The inner-product consists of multiplications and additions. The large number of binary multipliers and adders makes it nearly impossible to deploy the entire large DCNN such as AlexNet on embedded systems with a limited hardware resources and power budgets, not to mention more advanced DCNNs, such as VGG [40] and ZFNet [41] with even more neurons.

The SC technique, on the other hand, can potentially overcome this limitation and achieve a drastically smaller hardware footprint and higher energy efficiency. SC-DCNN [1] performs design space explorations on SC-based DCNNs for LeNet-5. However, it lacks the design and optimization at both function block level (e.g., ReLU or APC-based inner product block) and overall DCNN level, and result in a notable degradation in application-level accuracy because of the usage of tanh activation function. In order to overcome these limitations, the ReLU function block needs to be designed in SC domain and avoid the degradation in application-level accuracy. Even for the inner-product block which has already been investigated in SC-DCNN, it needs further optimizations to satisfy the requirements of energy efficiency, performance and accuracy. Moreover, an overall design optimization is necessary in order to optimize the overall energy efficiency while satisfying the application-level accuracy of DCNN.

In the following subsections, we introduce the ReLU function block design and the inner-product function block design in order to address the aforementioned drawbacks of SC-DCNN. And we also propose the optimization on the overall DCNN architecture including weight storage optimization, co-optimization on the feature extraction blocks, and pipelining-based optimization.

B. ReLU Function Block Design

ReLU has become the most popular activation function in state-of-the-art DCNNs, however, only hyperbolic tangent/sigmoid functions have been implemented in the SC domain in previous works [31,42]. Therefore, it is important to have the design of SC-based ReLU block in order to accommodate the SC technique in the state-of-the-art large-scale DCNNs, such as AlexNet for ImageNet applications.

The mathematical expression of ReLU is $f(x) = max(0, x)$, i.e., when input x is less than 0, the activation result is 0, otherwise the activation result is x itself. This characteristic of ReLU gives rise to a challenge for SC-based designs. Since x is represented by a stochastic bit-stream in SC with length m, we can only intuitively determine its sign and value through a counter using m clock cycles. This straightforward implementation of ReLU function in SC domain undoubtedly leads to a significant extra delay and energy overhead. On the other hand, the bit-stream-based representation in SC restricts the number it represents within the range [-1, 1], and as a result, the output of SC-based ReLU block should be clipped to 1. The clipped ReLU in the SC domain is expressed as $f(x) = min(max(0, x), 1)$.

Four concerns should be addressed to develop an effective SC-based ReLU block for DCNN applications: *i)* the application-level accuracy of the overall DCNN should be high enough if the ReLU activation result is clipped to 1; *ii)* determining whether the input x is a negative number without causing extra latency; *iii)* generating of SC bit-stream representing zero when the input x is less than zero; and *iv)* output x itself when $x \in [0, 1]$. In this section, the design of SC-based ReLU block is presented to resolve these concerns.

The premise that the SC-based ReLU block can be adopted in DCNNs is that the clipped ReLU would not bring about significant application-level accuracy degradation. Accordingly, we perform a series of experiments on representative DCNNs LeNet-5 and AlexNet by replacing their activation functions with the clipped ReLU. According to the experiment results, for AlexNet with ImageNet dataset [43], the clipped ReLU causes no significant accuracy degradation for the overall DCNN whereas for LeNet-5 with MNIST dataset [44], clipped ReLU even improve the accuracy by more than 0.1%. Therefore, the clipped ReLU is appropriate for the state-of-the-art DCNNs. This addresses the first concern.

To avoid extra latency, the sign of the number represented by the bit-stream should be estimated dynamically and synchronously. The SC-based ReLU proposed in this work implements the dynamic estimation by accumulating

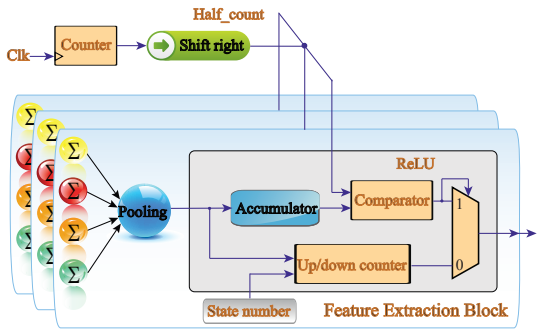


Fig. 6. **Diagram of the proposed ReLU block.**

the bit-stream and comparing the accumulated value with a reference number. Since in a stochastic bit-stream, the 1's are randomly distributed, the number represented by a bit-segment is approximately equal to the number represented by the whole bit-stream. For instance, when 0.5 is represented by a 1024-bit bit-stream, we consider both the first-half (512-bit) and the second-half bit-streams are approximately equal to 0.5. Consequently, by accumulating the bit-stream, the number represented by the accumulated number will asymptotically converge to the actual number represented by the whole bit-stream. On the other side, under the context of bipolar representation, the number zero is presented by a bit-stream with 50% of 1's, as $(0+1)/2 = 0.5$. Therefore, if the accumulated number is less than half of the clock cycles for accumulation, the number represented by the bit-stream is (likely to be) less than 0. The SC-based ReLU block outputs a bit of 1 to enforce the output to equal zero by increasing the number of 1's. The second and the third concerns are addressed by this accumulating and dynamic comparison strategy.

Similarly, if the accumulated number is greater than half of the clock cycles for accumulation, the number represented by the bit-stream is (likely to be) greater than 0. The current output of the SC-based ReLU is determined by the output of the finite state machine, which is homogeneous with Btanh. The last concern is addressed as well.

Figure 6 illustrates the proposed architecture of SC-based ReLU. The input of SC-based ReLU is accumulated, and the accumulation result is compared with a reference number (half of the passed clock cycles). The comparator output is used as an input and also the control signal of the multiplexer. If the accumulation result is less than the reference number, the comparator outputs a 1 and is selected by the multiplexer as the output of SC-based ReLU block. Otherwise, the output is determined by the FSM inside the SC-based ReLU block. Please note that the proposed SC-based ReLU will not incur any extra latency.

The algorithm of the proposed SC-based ReLU is illustrated in Algorithm 1. Please note that the $Positive$ signal is used to adjust the SC-based ReLU for different types of APCs. When the outputs of APCs represent the number of 1's among inputs, the normal logic is assigned to the output of SC-based ReLU. When the outputs of APCs represent the number of 0's, the inverted logic is assigned. The purpose is to make the output of SC-based ReLU (and thereby the whole FEB) not affected by the types of APCs.

Figure 7 shows the MATLAB simulation results of the proposed SC-based ReLU using different bit-stream lengths, and the simulation curves of the clipped-ReLU are also depicted.

Algorithm 1: Proposed SC-based ReLU hardware

input : $BitMatrix$ is the output of the previous pooling block
each column of the matrix is a binary vector
$Cycle_{half}$ is the half of the passed clock cycles
S is the FSM state number
N is the input size of a feature extraction block
m is the length of a stochastic bit-stream
$Positive$ indicates whether APC's output represents the number of 1's
output: Z is a bit-stream output by ReLU
$S_{max} = S$; //upper bound of the state
$S_{half} = S/2$;
$State = S_{half}$; //$State$ is used to record the state history
$Accumulated = 0$; //to accumulate each column of $BitMatrix$
if $Positive == 1$ **then**
  $ActiveBit = 1$; $InactiveBit = 0$;
else
  $ActiveBit = 0$; $InactiveBit = 1$;
for $i++ < m$ **do**
  $BinaryVec = BitMatrix[:i]$; //current column
  $State = State + BinaryVec * 2 - N$; //update current state
  //accumulate current column of the input
  $Accumulated = Accumulated + BinaryVec$;
  if $Accumulated < Cycle_{half}$ **then**
    $Z[i] = ActiveBit$;
    //enforce the output of ReLU to be greater than or equal to 0,
    //otherwise the output is determined by the following FSM
  else
    if $State > S_{max}$ **then**
      $State = S_{max}$;
    else
      if $State < 0$ **then**
        $State = 0$;
    if $State < S_{half}$ **then**
      $Z[i] = ActiveBit$;
    else
      $Z[i] = InactiveBit$;

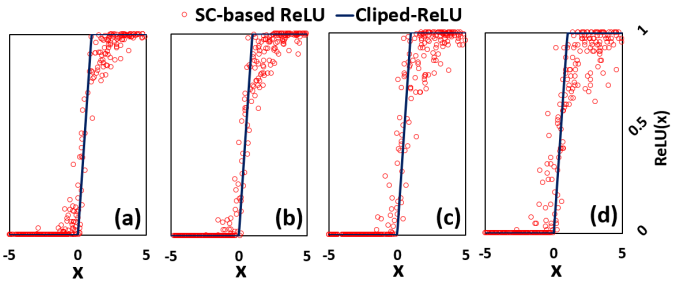


Fig. 7. **Results of the proposed SC-based ReLU using different bit-stream length (a) 1024, (b) 512, (c) 256, (d) 128.**

We randomly generate 1000 numbers for each experiments to test the SC-Based ReLU accuracy. As each bit is processed in one clock cycle, the $Cycle_{half}$ in Algorithm 1 represents the half of the number of passed bits. In this simulation, we set $Positive = 1$ to count the number of ones in the bit-stream. The average inaccuracies (the difference between the clipped-ReLU and the SC-based ReLU) of using 1024-bit length and 128-bit length are 0.031 and 0.057, respectively. We can conclude that SC-based ReLU can guarantee a high accuracy in DCNNs.

C. Inner-Product Block Optimization

We optimize the APC-based inner-product block with a potentially large number of inputs. Inner-product calculates the "summation of products" and involves both multiplication and

addition operations. Hence, we optimize both multiplication and APC-based addition in SC.

1) Transmission Gate Based Multiplication: As discussed before, the multiplications are implemented with XNOR gates in bipolar SC. Generally, an XNOR gate costs at least sixteen transistors if it is implemented in static CMOS technology, and its simplest structure in gate-level is shown in Figure 8 (a). However, if the XNOR gate is implemented with transmission gates, only eight transistors are needed, leading to 50% savings in hardware. The main drawback of potential voltage degradation of a transmission gate does not cause latent errors for three reasons: *i)* the multiplication operations are only performed in the first sub-layer of each network layer, so any latent voltage degradation will not be significant; *ii)* the following APCs and activation blocks are implemented with static CMOS technology, so any minor voltage degradation introduced by transimission gates will be compensated; *iii)* SC itself is soft error resilient, i.e., a soft error at one single bit has a negligible impact on the whole bit-stream. The structure of the transmission gate based XNOR gate is illustrated in Figure 8 (b).

2) APC Optimization: Approximate Parallel Counter (APC) [34] has been designed for efficiently performing addition with a large number of inputs in SC domain. More specifically, it efficiently counts the total number of 1's in each "column" of the input stochastic bit-streams and the output is represented by a binary number, as shown in Figure 4 (c). The APC consists of two parts: approximate units (AU), implemented by a combination of simple two-input gates such as AND/OR gate, and an accurate Parallel Counter (PC) with size significantly reduced. The PC circuit consists of a network of full adders for precisely counting the total number of 1's among the input bit-streams. Although the literature [34] presented the operation principle of APC, there is no existing work targeting at optimization of the performance and energy efficiency. We mitigate this limitation by presenting a holistic optimization framework of APC in the following.

First, we investigate the design optimization of adder trees in PC to refine APC design. A conventional PC uses full adders and half adders to calculate the number of active inputs (the total number of 1's). Each adder reduces a set of three inputs (for full adder) or two inputs (for half adder) with weight 2^n into an output line with weight 2^n and another output with weight 2^{n+1}, which correspond to the summation and output carry, respectively. To reduce the area and power & energy consumption of APC, we design adder tree using *inverse mirror full adders* [45], i.e. mirror full adders without output inverters, whose outputs are the logical inversion of summation and carry out bits. Compared to a full adder synthesis results (from Synopsys Design Compiler) requiring 32 transistors, an inverse mirror full adder only costs 24 transistors. An

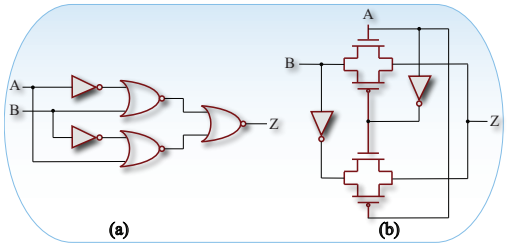


Fig. 8. **XNOR gate implementations. (a) Static CMOS design, (b) Transmission gate design.**

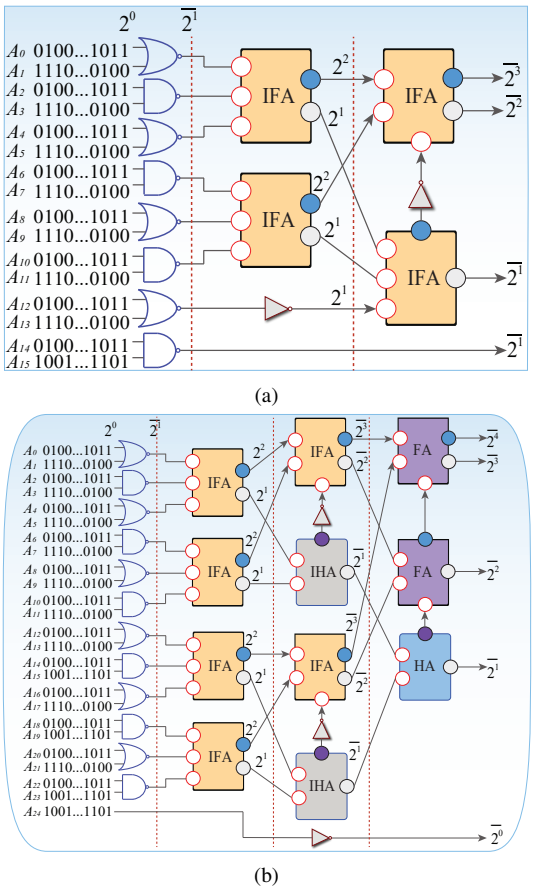


Fig. 9. **(a) Proposed 16-input APC structure, (b) Proposed 25-input APC structure.**

adder tree design is available for the PC using inverse full adders, in which the odd layer (of adders) outputs the inverse values of summation and output carry, representing the number of inactive inputs (the total number of 0's). The results are inverted back in the subsequent even layer of adders. Inspired by the same idea of using inverse logic, NAND/NOR gates can be used to construct the AU layer instead of AND/OR gates, to achieve further delay/area reductions.

Depending on the input size, the output of the proposed APC can either represent the number of 1's among the input bit-streams, or the number of 0's. Please note that the activation function needs to be modified if the APC output represents the number of 0's as discussed in the ReLU block design. As an example, the proposed 16-input APC design is shown in Figure 9 (a).

Next, we discuss the APC designs for input size that is not a power of two. An example of the proposed 25-input APC is shown in Figure 9 (b). Two modifications are needed compared with the previous case. First, *arithmetic inverse half adders* are required to calculate the number of inactive inputs (number of 0's among inputs). In addition, in this case, the final output of APC should be the non-inverted value compared with the inputs to the adder tree. In other words, if the inputs of adder tree represent the number of 0's (inactive inputs), then the APC output must also be the number of 0's. The reason is as follows: the summation of the number of 0's and the number of 1's should be equal to the input size (e.g., 25 as shown in Figure 9 (b)), whereas the inverse operation in adders assumes that their summation is 2^{N+1}, where N is the number of bits in the output binary number. Thus, the final layer of adder

TABLE I
Comparison of inner-product blocks before and after optimization using 1024-bit-stream.

Input Size	Approach	Optimization	Delay(ns)	Area(μm^2)	Energy (fJ)
16	SC	before	0.57	51.1	26.2
		after	0.49	26.6	22.8
	binary	-	2.02	2759.4	4775.4
32	SC	before	0.88	134.3	133.9
		after	0.78	82.7	122.3
	binary	-	2.15	5589.7	10618.2
64	SC	before	1.24	253.5	328.3
		after	1.12	147.1	294.3
	binary	-	2.38	11279.9	24095.1
128	SC	before	1.46	597.4	1069.7
		after	1.32	380.9	996.2
	binary	-	2.61	22664.7	53492.0
256	SC	before	1.78	1177.6	2652.3
		after	1.62	740.3	2450.6
	binary	-	2.84	45438.7	117201.1

tree should use either adders or inverse adders to generate non-inverted results compared with the inputs.

Table I shows the comparison of inner-product blocks before and after optimization using the 1024-bit-stream. After applying the optimization on the inner-product blocks, the hardware performance in terms of clock period, area, and energy are all reduced, especially the area. Table I also demonstrates the advantages of SC over conventional binary computing. We can observe that the SC delay/area/energy are much smaller than binary's, this is because SC based inner-product blocks taking multiple input bit-streams in a parallel manner with simple gate logic, while the binary logic compute equivalent binary numbers bit by bit with complex gate logic.

D. Weight Storage Optimization

The main computing task of an inner-product block is to calculate the inner-products of x_i's and w_i's. x_i's are inputs of neurons, while w_i's are weights obtained during training, stored, and used in the hardware-based DCNNs. The number of weights is skyrocketing as the structure of DCNNs becomes much deeper and more complex. For example, LeNet-5 [3] includes 431k parameters, AlexNet [5] has around 61M parameters, and VGG-16 [40] contains over 138M parameters. It is urgent to explore the techniques to store the tremendous parameters efficiently. In convolutional layers, weights are shared within filter domain, while in fully connected layers, the number of weights is enormous and independent. Thus the weights need to be either shared or reduced. The reduction of weights has been explored in many previous works such as [20,46], however, weight sharing lacks the discussion. In this section, we present a simple weight reduction method and a clustering based weight sharing optimization. The methods presented can be combined with weight reduction/pruning methods in related works.

We use *Static random access memory (SRAM)* for weight storage due to its high reliability, high speed, and small area. The specifically optimized SRAM placement schemes and weight storage methods are imperative for further reductions of area and power (energy) consumptions. In general, DCNN will be trained with single floating point precision. Thus on hardware, up to 64-bit SRAM is needed for storing one weight value in the fixed point format to maintain its original high precision. This scheme can provide high accuracy as there is almost no information loss of weights. However, it also brings about high hardware consumptions in that the size of SRAM and its related read/write circuits is increasing with the increasing of precision of the stored weight values.

According to our software-level experiments, many least significant bits far from the decimal point only have a very limited impact on the overall application-level accuracy, thus the number of bits for weight representation in the SRAM block can be significantly reduced. We adopt a mapping equation that converts a weight in the real number format to the binary number stored in SRAM to eliminate the proper numbers of least significant bits. Suppose the weight value is x, and the number of bits to store a weight value in SRAM is w (which is defined as the *precision* of the represented weight value in this paper), then the binary number to be stored for representing x is:

$$y = \frac{Int(\frac{x+1}{2} \times 2^w)}{2^w} \tag{2}$$

where $Int()$ means only keeping the integer part. Please note that the binary numbers stored in SRAMs are fed into efficient *Random Number Generators (RNGs*) to generate stochastic numbers at runtime. For instance, a 6-bit binary number can be used to generate a stochastic number with 1024-bit length through RNG. Hence, there is no need to store the entire 1024 bit stochastic number in SRAM. The overhead of RNGs is also taken into account in our experiments. Therefore, this weight storage method can significantly reduce the size of SRAMs and their read/write circuits through decreasing the precision. The area saving achieved by this method based on estimations from CACTI 5.3 [47] is 10.3×.

Weight Clustering. As mentioned before, a state-of-the-art DCNN contains millions of weights. A large amount of SRAM will be consumed for storing all these weights. In fact, many weight values can be rounded to a neighboring value without significant accuracy loss according to our experiments. Therefore, we investigate the k-means based weight clustering method that clusters all weights into clusters and rounds the weights in each cluster to one centroid value. Consequently, only a part of weight values need to be stored in SRAM. A multiplexer is used to select a weight from a SRAM block for each w_i of an inner product block, and the selection signals are stored in SRAM block as well. Suppose the filter size is $p \times p$, each weight occupies n bits, and storing one bit consumes t units hardware resources on average (including read/write circuits). Accordingly, the size of an SRAM block before clustering is $p^2 \times n \times t$. After clustering, only s weights are needed, thus the size of an SRAM block for storing weights is $s \times n \times t$. Since an inner product block has p^2 weight values, p^2 multiplexers are required for each inner product block, and $p^2 \times log_2 s \times t$ units hardware resources are needed for storing the selection signals. Suppose the size of a multiplexer is m units hardware resources, and there are q inner product blocks for extracting a feature map. The area saving achieved by the clustering method is $p^2 \times n \times t - (s \times n \times t + p^2 \times log_2 s \times t + p^2 \times m \times q)$ for each feature map.

As shown in Figure 10(a), when the clustering is performed on all weights of the network, the application-level error rate vibrates obviously with the change of the clustering number, and the error rates in many cases exceed 10%. It indicates that the clustering on all weights is not practicable.

Then we perform the clustering on weights within each single layer to explore the application-level accuracy performance. As illustrated in Figure 10(b), when the clustering is performed on each layer from Conv1 to FC2, desirable

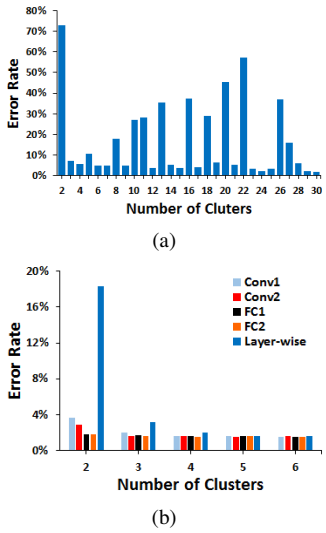


Fig. 10. **Application-level error rates for (a) clustering through all layers, (b) clustering within each layer and layer-wise clustering.**

application-level accuracy can be obtained while the number of clusters is more than three. Inspired by the experimental results, we investigate the application-level accuracy when the clustering is performed on the whole network but each layer is individually clustered (called *layer-wise*, and different layers may have the different number of clusters). When all layers are individually clustered into five or more clusters, the application-level error rate is less than 2%.

E. Optimization on Feature Extraction Block and the Overall DCNN

1) Co-optimization on FEBs: In the FEB, the inner-product, max pooling and ReLU function blocks are connected in series, and the imprecision of one function block will be propagated to the subsequent block(s) within this FEB. Considering the intra-FEB imprecision propagation effect due to the cascade connection, the parameters of the inner-product, max pooling and ReLU function blocks inside one FEB should be jointly optimized. The goal of co-optimization through the SC-based FEB is to approach the accuracy level of software FEB.

We propose an optimization function $S = f(N)$, where S and N denote the FSM state number in ReLU and the fan-in, respectively. First of all, given N, each inner-product block is optimized. Next, in order to derive the optimization function, we simulate each FEB with all the function blocks connected together, and select the S that yields the highest precision under a given N. Below is the empirical function that is extracted from comprehensive experiments obtaining the optimal state number providing a high precision:

$$S = f(N) \approx 2 \cdot N \tag{3}$$

Figure 11 shows the optimized FEB precision under different combinations of the input size and the bit-stream length. One can observe that the FEB can work with a short bit-stream length (i.e., 128 bits) without incurring significant accuracy degradation. Moreover, as a desirable effect, the accuracy will increase with the increase of the input number, because the imprecisions tend to mitigate each other with the input size increase. Table II summarizes the hardware performance of FEBs with the different input sizes when the bit-stream

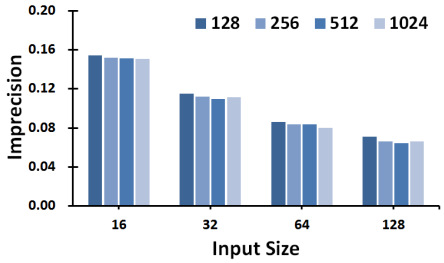


Fig. 11. **Optimized FEB precision vs input size under different bit-stream lengths.**

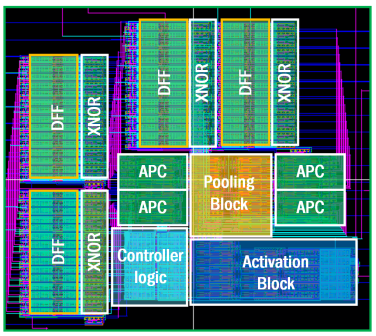


Fig. 12. **Layout of a 64-input FEB using the proposed APC, pooling and activation blocks.**

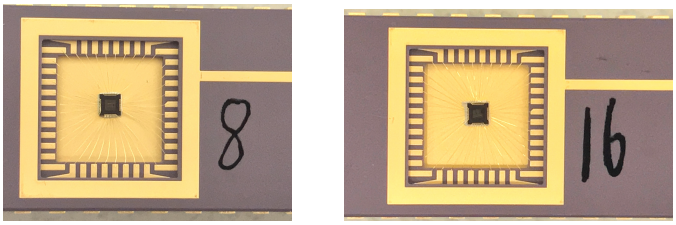
length is 1024, which shows a sub-linear growth in terms of area/power/and energy with the increase of input size.

Using the optimization function, we derive the optimal configuration of a 64-input FEB with four 16-input APCs and 4-to-1 pooling. The full customized layout design of this FEB using Cadence Virtuoso is shown in Figure 12. Note that multiple D flip-flop (DFF) arrays are used to temporarily hold inputs due to the limited I/O bandwidth of the foundry. Shown in Figure 13, we taped out the 8-bit and the 16-bit FEB as the proof-of-concept. We tested our chips using a Altera Cyclone V FPGA 14, random bit-streams are fed into the FEB chip, the results are displayed on an oscilloscope as Figure 15 shows.

2) Pipeline Based DCNN Optimizations: In this work, we propose a two-tier pipeline based network optimization for HEIF as shown in Figure 16. The first-tier pipeline is placed in between different convolutional and fully-connected layers, i.e., inserting DFFs between consecutive layers to hold the temporary results, which enables pipelining across the deep layers of DCNNs. The second-tier pipeline is placed within a layer which is inspired by [48]. More specifically, based on the delay results of inner product, pooling and ReLU blocks, we insert DFFs between the pooling unit and ReLU block in order to further reduce the system clock period. We place the pooling unit in the first stage. Because after pooling, the output size is reduced so that we can use less DFFs to save area & power & energy. To show the effectiveness of pipelining within

TABLE II
Hardware performance of FEBs with the different input sizes using 1024-bit-stream w/ and w/o pipeline based optimization

Optimization	Pipelining				Non-pipelining			
Input size	16	32	64	128	16	32	64	128
Clock Period (ns)	1.74	1.82	2.08	2.16	2.2	2.51	2.67	2.79
Area (μm^2)	910.8	1162.4	1569.4	2305.2	904.4	1102.3	1453.9	2149.5
Power (μW)	556.6	771.2	928.4	1409.4	421.5	490.3	659.9	973.5
Energy (fJ)	968.4	1403.5	1931.0	3044.2	927.3	1230.8	1762.0	2716.1



(a) (b)

Fig. 13. **FEB tape-out (a) 8-bit chip, (b) 16-bit chip.**

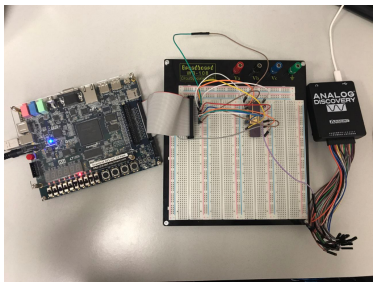


Fig. 14. **Testing platform for the fabricated chips**

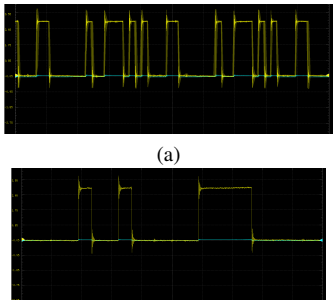
a layer, we also evaluate the hardware costs for FEBs without pipelining in the right section in Table II. Comparing the results in Table II, we observe that the pipelining optimization significantly reduce the delay (clock period) by about 22% in average with slight area & power & energy increase by DFFs.

An additional key optimization knob is the bit-stream length. A smaller bit-stream length in SC can almost improve the energy efficiency in a proportional manner. However, we must ensure that the overall application-level accuracy is maintained when the bit-stream length is reduced, and therefore, a joint optimization is required. In this procedure, we first optimize the accuracy of each function block, i.e., APC, max pooling and ReLU, to reduce the imprecision within an FEB. Furthermore, we conduct co-optimization through FEB to find the best configuration of each unit inside one FEB, in order to mitigate the propagation of imprecision and maintain the overall application-level accuracy.

IV. Results

The proposed HEIF is to accelerate Deep Convolutional Neural Networks (DCNNs). Besides,it is applicable to various deep models such as Deep Belief Networks (DBNs), Long Short-Term Memory (LSTM), etc., where similar computations are conducted. In this section, to demonstrate the effectiveness of the proposed HEIF, we perform thorough optimizations on two widely used DCNNs as examples, i.e., LeNet-5 [49] and AlexNet [5], to minimize area and power (energy) consumption while maintaining a high application-level accuracy. The feature extraction blocks, the pipeline, the bit-stream length, and the weight storage schemes are carefully selected/optimized in the procedure.

The LeNet-5 is a widely used DCNN structure with a configuration of 784-11520-2880-3200-800-500-10. The MNIST handwritten digit image dataset [50] is used to evaluate LeNet-5, which consists of 60,000 training data and 10,000 testing data. The AlexNet, on the other hand, is a much larger DCNN with a configuration of 290400-186624-64896-64896-43264-4096-4096-1000. The accuracy of AlexNet is measured on the ImageNet dataset (ILSVRC2012)[43], which contains



(a)

(b)

Fig. 15. **Tested Waveform for (a) 8-bit chip, (b) 16-bit chip.**

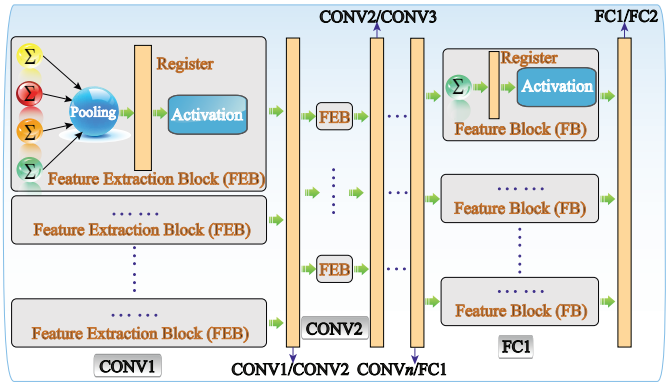


Fig. 16. **Two-tier pipeline design in the HEIF framework.**

$1.28M$ training images, $50k$ validation images, and $100k$ test images with 1,000 class labels.. The delay, power and energy of FEB are obtained from synthesized RTL under Nangate 45nm process [51]. The key peripheral circuitry in the SC domain, e.g., the random number generators, are developed using the design in [42] and synthesized using Synopsys Design Compiler, whereas the SRAM blocks are estimated using CACTI 5.3 [47].

Table III concludes the performance and hardware cost of the proposed HEIF on LeNet-5 implementation. One can observe that the proposed HEIF can realize the entire LeNet-5 with only 0.10% accuracy degradation compared to the software accuracy of our software-based implementations. Table IV compares the performance and hardware cost of the proposed HEIF with the existing hardware platforms on the MNIST dataset. It can be observed that compared with the other platforms, the proposed HEIF yields the highest throughput, area efficiency and energy efficiency while approaching the highest software accuracy, i.e. 99.77%, demonstrating the effectiveness of the SC technology and our proposed holistic optimization procedure. Compared with the high-performance version of SC-DCNN in [1], the proposed method achieves up to 0.81% accuracy increase, and $4.1\times$, $6.5\times$ and $5.5\times$ improvement in terms of throughput, area efficiency and energy efficiency, respectively. Compared with the low-power version of SC-DCNN, the proposed method achieves improved accuracy due to the overall optimization on the cascade connection of function blocks and the novel ReLU design, whereas the area, power and energy efficiency gain are mainly achieved through APC optimization, pipelining technique, bit-stream length reduction, and weight storage optimization.

Next we present the results of HEIF on the large-scale AlexNet applications. We trained AlexNet using ImageNet

TABLE III
Application-level performance and hardware cost of LeNet-5 implementation using the proposed HEIF.

Bit Stream	ReLU		ReLU Clipped		Area (mm^2)	Power (W)	Delay (ns)	Energy (μJ)
	Validation	Test	Validation	Test				
1024	1.10%	0.99%	1.07%	0.88%	22.9	2.6	2498.6	6.4
512	1.09%	0.98%	1.12%	0.87%			1249.3	3.2
256	1.12%	1.00%	1.13%	0.91%			624.6	1.6
128	1.08%	1.01%	1.18%	0.93%			312.3	0.8
software	1.09%	0.94%	0.94%	0.83%			-	
highest software accuracy in the literature [52]				0.23%			-	

training set by our own configurations. To follow the stochastic computing paradigm, we use scaled pixel values within [0,1] instead of original range [0,255]. Because data pre-processing first deducts the mean value of each image from each pixel value, the input then ranges in [-1,1]. Moreover, we use clipped ReLU to restrain the activation output to be [0,1]. We also move pooling units before ReLU so that we can save resource of ReLU in the aspect of hardware cost. The trained network achieves top-1 and top-5 accuracies of 56.56% and 80.48% on the test set, respectively. To the best of our knowledge, the existing hardware platforms either implemented one computation layer of the AlexNet [20], built a reconfigurable circuit to accelerate each layer separately [22], or designed a reconfigurable system that can be connected in a chip system to deal with large computation tasks [21]. Table V lists the existing hardware platforms for AlexNet implementation. As EIE [20] provided the results on the fully-connected FC7 layer of AlexNet, we evaluate the proposed HEIF on the same FC7 layer of AlexNet. We apply the same weight compression technique in [55], making a fair comparison. Note that Table V is a list of existing platforms instead of a strict comparison table, because the implementation scales and method of different works are not the same (and some are not discussed in details in papers). One can observe from Table V that the proposed HEIF has the smallest footprint due to the small footprint of each stochastic computing component, and achieves the best performance in terms of throughput, area efficiency and energy efficiency.

Finally, we investigate the capacity of HEIF on implementing each layer and the full AlexNet. We evaluate the hardware performance of each layer in AlexNet separately and conclude the area, power and layer delay in Table VI. Table VI also concludes the accuracy performance of the proposed HEIF on the full AlexNet. It is observed that the proposed HEIF can realize the entire AlexNet with only 1.35% top-1 accuracy degradation and 1.02% top-5 accuracy degradation compared to the software accuracy of our software-based implementations. As shown in Table VI, the convolution layer Conv5 and fully-connected layers FC6-FC8 can be implemented using the proposed HEIF efficiently. However, one should note that due to a large number of neurons in convolution layers Conv1-Conv4, the area and power consumptions of these layers are significant. Hence to make tape-out possible, we have to adopt a reconfigurable approach to implement the large layers in a time-multiplexed manner, which is also a future extension of this work.

V. Discussion

A. Scalability

The proposed SC paradigm is able to process the computation as the (convolutional) neural network architecture gets deeper with the help of pipelining. Since the input size in each inner-product function block in convolutional layers is the corresponding filter size, the key challenge the SC-based components face is the booming of inputs for each inner-product function block in the fully-connected layers.

The experimental results show that a 4096-input inner-product function block consumes power as high as 6.2mW and delay is 3.3ns which is longer than smaller blocks. Meanwhile, it is as big as $11,973\mu\text{m}^2$ and needs 20.64pJ to drive a large APC. Considering in AlexNet, FC7 layer contains 4096 inner-product function blocks, the concurrent circuit with such power & energy consumption is not achievable. Thus the model must be compressed to reduce the input size in FC layers. We applied compressed model mentioned in [55], the input size of each neuron is pruned to as low as 9% of the original number. The path delay is then improved by 50% because of the shorter path along hierarchical adder in APC. And the power and energy are reduced to 0.9mW and 6.3pJ, respectively, while area efficiency is improved by $9\times$.

With the compressed design of inner-product function block, we can scale the SC-based framework to the state-of-the-art large-scale DCNNs, considering that the computation within those DCNNs is covered by our framework. Some special normalization layers such as *Local Response Normalization (LRN)* and regularization layers such as *Dropout* are the competition-directed optimization, which can be removed with a slight sacrifice of accuracy [5,56] to improve the overall efficiency of the network. These non-resource-exhausting operations are the next step to fully design a general SC-based framework for DCNN which is also the future work for other hardware acceleration researches for DCNNs.

B. Energy Efficiency

SC-based design has achieved high energy efficiency which is shown in Table IV and Table V. However, the consumed energy is proportional to the stage delay of the network and the length of bit-streams. Since bit-streams are processed sequentially in the network and the hardware building blocks are given, reducing the length of bit-streams can efficiently reduce the energy consumption without increasing the power. This is a key characteristic of SC as long as the overall accuracy satisfies certain constraints. Shown in Table III, when the bit-stream length is reduced to 128, compared with a bit-stream length of 1024, the energy efficiency is increased by $8\times$ with only 0.11% validation application-level accuracy loss and 0.05% test accuracy loss. Meanwhile, the footprint and power are not increased, for the hardware is not modified. There is a potential for a shorter bit-stream and much less energy which is due to the trade-off between accuracy and energy efficiency. Note that the energy & power related results are the synthesis results using Synopsys Design Compiler, the power dissipation on the clock tree is neglected although that on the sequential elements (DFFs) is already accounted for. Compared with binary-based designs, SC-based designs (e.g., the proposed HEIF) do not contain a large number of sequential elements because of the sequential processing nature. Also, the operating frequency of the proposed HEIF (410MHz) is not overly high. Therefore, the energy dissipation induced by the clock tree will not be very significant.

C. Application-level Accuracy

The proposed highly-efficient SC-based framework ensures high application-level accuracy of DCNN. Taking LeNet and

TABLE IV
Comparison with existing hardware platforms for handwritten digit recognition using the MNIST [50] dataset

Platform	Network Type	Year	Platform Type	Clock (MHz)	Area (mm^2)	Power (W)	Accuracy (%)	Throughput (Images/s)	Area Efficiency (Images/s/mm^2)	Energy Efficiency (Images/J)
2×Intel Xeon W5580	CNN	2009	CPU	3200	263	156	99.17	656	2.5	4.2
Nvidia Tesla C2075	CNN	2011	GPU	1150	520	202.5	99.17	2333	4.5	3.2
Minitaur [53]	ANN[1]	2014	FPGA	400	N/A	≤1.5	92.00	4880	N/A	≥3253
SpiNNaker [54]	DBN	2015	ARM	150	N/A	0.3	95.00	50	N/A	166.7
TrueNorth [46]	SNN[2]	2015	ASIC	Async	430	0.18	99.42	1000	2.3	9259
SC-DCNN (No.6)[1]	CNN	2016	ASIC	200	36.4	3.53	98.26	781250	21439	221287
SC-DCNN (No.11)[1]	CNN	2016	ASIC	200	17.0	1.53	96.64	781250	45946	510734
HEIF(128bit)	CNN	2016	ASIC	410	22.9	2.6	99.07	3203125	139874	1231971

[1]ANN: Artificial Neural Network; [2]SNN: Spiking Neural Network

TABLE V
List of existing hardware platforms for image classification using (part of) the AlexNet [5] on ImageNet [43] dataset

Platform	Year	Platform Type	Memory Type	Area (mm^2)	Power (W)	Throughput (Images/s)	Area Efficiency (Images/s/mm^2)	Energy Efficiency (Images/J)
2×Intel Xeon W5580	2009	CPU	DRAM	263	156	139	0.5	0.9
Nvidia Tesla C2075	2011	GPU	DRAM	520	202.5	573	1.1	2.8
DaDianNao [21]	2014	ASIC	eDRAM	67.7	15.97	147938	2185	9263
Eyeriss [22]	2016	ASIC	DRAM	12.25	0.28	35	2.8	125
EIE-64PE [20]	2016	ASIC	SRAM	40.8	0.59	81967	2009	138927
EIE-256PE [20]	2016	ASIC	SRAM	63.8	2.36	426230	6681	180606
HEIF(128bit)	2016	ASIC	SRAM	24.7	1.9	2520161	102030	1326400

TABLE VI
Hardware cost and performance of the whole AlexNet implementation using the proposed HEIF

Layer	Layer Type	Area(mm^2)	Power(W)	Delay(ns)
Conv1	Conv-Max-ReLU	366.2	66.4	3.0
Conv2	Conv-Max-ReLU	116.2	20.0	3.1
Conv3	Conv-Max-ReLU	131.9	23.0	2.7
Conv4	Conv-Max-ReLU	131.1	23.0	2.7
Conv5	Conv-Max-ReLU	20.1	3.3	2.7
FC6	FC-dropout	45.1	2.0	2.1
FC7	FC-dropout	24.7	1.9	2.1
FC8	FC-softmax	12.8	0.5	2.1
Total	-	848.1	140.2	-
Top-1 accuracy	software: 56.56%		HEIF: 55.21%	
Top-5 accuracy	software: 80.48%		HEIF: 79.46%	

AlexNet as examples for DCNNs, shown in Table III and Table VI, the proposed framework can achieve as high as 99.07% test accuracy which outperforms the previous SC-based related work [1] on LeNet-5. Please note the trained software model for LeNet in this work is able to achieve 99.17% test accuracy, which means the HEIF only downgrades 0.1% accuracy to achieve much higher energy efficiency. Moreover, in the large-scale application of ImageNet classification of 1,000 labels, using AlexNet, the proposed framework can achieve as high as 79.46% top-5 accuracy which is only 1.02% performance degradation from the trained model.

This is because the combination of DCNN and SC paradigm along with the proposed optimization framework mitigates the errors brought by the imprecision of each function block. In LeNet-5, an FEB takes 25 inputs which shows an imprecision of 0.11, and the fully-connected neuron causes an imprecision of 0.06. Similarly, in AlexNet, an FEB taking 121, 25, 9 inputs gives imprecision of 0.07, 0.11, 0.18 respectively. Interestingly, when translating the hardware-level imprecision to application-level accuracy, the latter is not downgraded significantly, with only 0.1% test accuracy loss in LeNet and 1% top-5 accuracy in AlexNet. This is because (i) the imprecisions can be both positive or negative and can mitigate each other when the input size is large, and can be mitigated in the pooling block and by the scaling function of inner products, and (ii) random and small deviations of hardware results will not significantly affect the software classification results. The theoretical analysis and quantitative proof of translating hardware-level imprecisions into application-level errors will be another promising direction of SC research and the more general research area of approximate computing.

VI. Related Works

References [5,10,57,58] leveraged the parallel computing and storage resources in GPUs to efficiently implement DCNNs. FPGA-based accelerators are another attractive option for the hardware implementation of DCNNs [12,13] due to its programmability, the high degree of parallelism and short develop period. However, the current GPU- and FPGA-based implementations still exhibit a large margin of performance enhancement and power reduction. This is because (i) GPUs and FPGAs are general-purpose computing devices not specifically optimized for executing DCNNs; and (ii) the relatively limited signal routing resources in such general platforms restrict the performance of DCNNs which require high inter-neuron communication.

Alternatively, ASIC-based implementations of DCNNs have been recently exploited to overcome the limitations of general-purpose computing approaches. Three representative state-of-the-art works on ASIC-based implementations are Eyeriss [22], EIE [20], and the DianNao family, including DianNao [59], DaDianNao [21], ShiDianNao [60], and PuDianNao [61]. Eyeriss [22] is an energy-efficient reconfigurable accelerator for the large CNNs with various shapes. EIE [20] focuses specifically on the fully-connected layers of DCNN and

achieves high throughput and energy efficiency. The DianNao family [59]-[61] is the series of hardware accelerators designed for a variety of machine learning tasks (especially the large-scale DCNNs) with a special emphasis on the impact of memory on accelerator design, performance, and energy.

To provide the high energy efficiency and low hardware footprint required in embedded and portable devices, novel computing paradigms are needed. SC-based design of neural networks has been shown an attractive candidate to meet the stringent requirements and facilitate the widespread of DCNNs in low-power personal, embedded, and autonomous systems. [33] utilized stochastic logic to implement a radial basis function-based neural network. [32] presented the neuron design with SC for deep belief network. The design space exploration of SC-based DCNNs is recently performed in [1] for LeNet-5. However, there is *no existing work* that *1)* optimizes energy efficiency without compromising application-level accuracy and *2)* investigates comprehensive design optimizations of SC-based DCNNs with a large scale (e.g. AlexNet with ImageNet-scale) and wide applications.

VII. Conclusion

In this paper, we present HEIF, a highly efficient SC-based inference framework of the large-scale deep convolutional neural networks, with broad applications on (but not limited to) both LeNet-5 and AlexNet, in order to achieve ultra-high energy efficiency and low area/hardware cost. In this framework, we re-design the Approximate Parallel Counter and optimize stochastic multiplication while proposing for the first time SC-based Rectified Linear Unit (ReLU) activation function to track with the recent advances in software models. A memory storage optimization method is investigated to store weights efficiently. Lastly, overall optimizations on the cascade connection of function blocks in DCNN, pipelining technique, and bit-stream length optimization are investigated in order to achieve maximum energy efficiency while maintaining application-level accuracy requirements. The proposed framework achieves very high energy efficiency of $1.2M$ Images/J and $1.3M$ Images/J, and high throughput of $3.2M$ Images/s and $2.5M$ Images/s, along with very small area of 22.9 mm^2 and 24.7 mm^2 on LeNet-5 and AlexNet respectively. HEIF outperforms previous SC-DCNN by the throughput of $4.1\times$, by area efficiency of up to $6.5\times$ and achieves up to $5.6\times$ energy improvement.

Acknowledgment

This work is funded in part by the seedling fund of DARPA SAGA program under FA8750-17-2-0021. Besides, this work is also supported by Natural Science Foundation of China (61133004, 61502019) and Spanish Gov. & European ERDF under TIN2010-21291-C02-01 and Consolider CSD2007-00050.

References

[1] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 405–418. ACM, 2017.

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

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[8] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[9] Bryan Catanzaro. Deep learning with cots hpc systems. 2013.

[10] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[11] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*. Citeseer, 2011.

[12] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170. ACM, 2015.

[13] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. Design space exploration of fpga-based deep convolutional neural networks. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 575–580. IEEE, 2016.

[14] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2, 2015.

[15] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[16] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[17] Frederic Maire, Luis Mejias, and Amanda Hodgson. A convolutional neural network for automatic analysis of aerial imagery. In *Digital lmage Computing: Techniques and Applications (DlCTA), 2014 International Conference on*, pages 1–8. IEEE, 2014.

[18] Kishore Reddy Konda, Achim Königs, Hannes Schulz, and Dirk Schulz. Real time interaction with mobile robots using hand gestures. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 177–178. ACM, 2012.

[19] Nils Y Hammerla, Shane Halloran, and Thomas Ploetz. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880*, 2016.

[20] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. *arXiv preprint arXiv:1602.01528*, 2016.

[21] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[22] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. 14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 262–263. IEEE, 2016.

[23] Joshua San Miguel and Natalie Enright Jerger. The anytime automaton. In *Proceedings of the International Symposium on Computer Architecture*, 2016.

[24] Joshua San Miguel, Jorge Albericio, Natalie Enright Jerger, and Aamer Jaleel. The bunker cache for spatio-value approximation. In *Proceedings of the International Symposium on Microarchitecture*, 2016.

[25] Divya Mahajan, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, and Hadi Esmaeilzadeh. Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration. In *Proceedings of the International Symposium on Computer Architecture*, 2016.

[26] Jongse Park, Emmanuel Amaro, Divya Mahajan, Bradley Thwaites, and Hadi Esmaeilzadeh. Axgames: Towards crowdsourcing quality target determination in approximate computing. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 623–636. ACM, 2016.

[27] Daniel Lustig, Geet Sethi, Margaret Martonosi, and Abhishek Bhattacharjee. Coatcheck: Verifying memory ordering at the hardware-os interface. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 233–247. ACM, 2016.

[28] Kaisheng Ma, Xueqing Li, Karthik Swaminathan, Yang Zheng, Shuangchen Li, Yongpan Liu, Yuan Xie, John Jack Sampson, and Vijaykrishnan Narayanan. Nonvolatile processor architectures: Efficient, reliable progress with unstable power. *IEEE Micro*, 36(3):72–83, 2016.

[29] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):92, 2013.

[30] Brian R Gaines. Stochastic computing systems. In *Advances in information systems science*, pages 37–172. Springer, 1969.

[31] Bradley D Brown and Howard C Card. Stochastic neural computation. i. computational elements. *IEEE Transactions on computers*, 50(9):891–905, 2001.

[32] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *Proceedings of the 53rd Annual Design Automation Conference*, page 124. ACM, 2016.

[33] Yuan Ji, Feng Ran, Cong Ma, and David J Lilja. A hardware implementation of a radial basis function neural network using stochastic logic. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 880–883. EDA Consortium, 2015.

[34] Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. Approximate de-randomizer for stochastic circuits. *Proc. ISOCC*, 2015.

[35] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[36] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[37] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[38] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[39] Peng Li and David J Lilja. Using stochastic computing to implement digital image processing algorithms. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 154–161. IEEE, 2011.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[41] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[42] Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. An energy-efficient random number generator for stochastic circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 256–261. IEEE, 2016.

[43] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[44] Yann LeCun, Corinna Cortes, and Christopher JC Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.

[45] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[46] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

[47] S Thoziyoor, N Muralimanohar, JH Ahn, and N Jouppi. Cacti 5.3. *HP Laboratories, Palo Alto, CA*, 2008.

[48] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the International Symposium on Computer Architecture*, 2016.

[49] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 2015.

[50] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[51] Nangate 45nm Open Library, Nangate Inc., 2009.

[52] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[53] Daniel Neil and Shih-Chii Liu. Minitaur, an event-driven fpga-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(12):2621–2628, 2014.

[54] Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[55] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[56] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

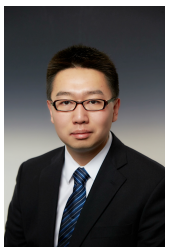
[57] Endre László, Péter Szolgay, and Zoltán Nagy. Analysis of a gpu based cnn implementation. In *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, pages 1–5. IEEE, 2012.

[58] GEORGE VALENTIN STOICA, RADU DOGARU, and CE Stoica. High performance cuda based cnn image processor, 2015.

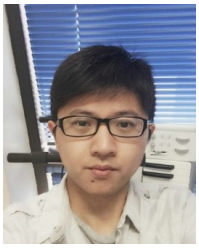
[59] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[60] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[61] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. Pudiannao: A polyvalent machine learning accelerator. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 369–381. ACM, 2015.



Zhe Li received his B.E. degree in Telecommunication Engineering from Beijing University of Posts and Telecommunications, China in 2012, and M.S. degree in Computer Engineering from Syracuse University, USA in 2014. He is currently working on his Ph.D. degree in Department of Electrical Engineering and Computer Science of Syracuse University. His research interests include deep learning applications and acceleration, neuromorphic computing and high performance computing.



Ji Li received the B.S. degree in microelectronics from Xian Jiaotong University in 2012, and the M.S. degree in electrical engineering from the University of Southern California in 2014. He is currently pursuing the Ph.D. degree in electrical engineering at the University of Southern California, under the supervision of Prof. Jeffrey Draper and Prof. Shahin Nazarian. His current research interests include resilient computing, neuromorphic computing, and the smart grid.

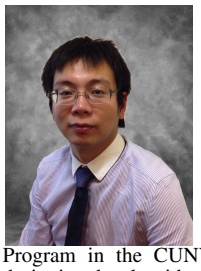
Ao Ren received his B.S. degree in Integrated Circuit Design and Integrated System from Dalian University of Technology in 2013, and he received his M.S. degree in Computer Engineering from Syracuse University in 2015. Presently he is a Ph.D. student under the supervision of Dr. Yanzhi Wang in Syracuse University, and his main research interest is the hardware acceleration for deep neural networks.

Ruizhe Cai received the B.S. degree in Integrated Circuit Design and Integrated System from Dalian University of Technology, Dalian, Liaoning, China, in 2014, and the M.S. degree in Computer Engineering from Syracuse University, Syracuse, NY, USA, in 2016. He was a research student in Communication and Computer Engineering at Tokyo Institute of Technology, Tokyo, Japan between 2013 to 2014. He is currently a Ph.D. student in Computer Engineering in Syracuse University, Syracuse, NY, USA. His research interests include neuromorphic computing, deep neural network acceleration, and low power design.

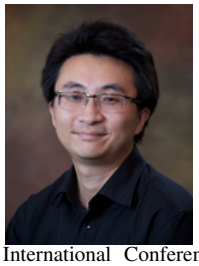
Caiwen Ding is currently a Ph.D. student in the department of Electrical Engineering and Computer Science at Syracuse University, NY, USA. His current research interests include high-performance and energy-efficient computing, hybrid electrical energy storage systems, and neuromorphic computing systems for hardware acceleration and cognitive frameworks.

Xuehai Qian is an assistant professor at the Ming Hsieh Department of Electrical Engineering and the Department of Computer Science at the University of Southern California. He has a Ph.D. from the Computer Science Department at University of Illinois at Urbana-Champaign. He has made several contribution to parallel computer architecture, including cache coherence for atomic block execution, memory consistency check, architectural support for deterministic record and replay. His recent research interests include system/architectural supports for graph processing, transactions for Non-Volatile Memory and acceleration of machine learning and graph processing using emerging technologies.

Jeffrey Draper received the BS degree in electrical engineering from Texas A&M University and the MSE and PhD degrees in computer engineering from the University of Texas at Austin. He holds a joint appointment as a research associate professor in the Ming Hsieh Department of Electrical Engineering and project leader at the Information Sciences Institute, University of Southern California. He has led the microarchitecture and/or VLSI effort on several large projects in the past 20 years, including many US Defense Advanced Research Projects Agency sponsored programs such as Integrity and Reliability in Integrated Circuits, Ubiquitous High-Performance Computing, Trust in Integrated Circuits, Radiation Hardening by Design, Polymorphous Computing Architectures, and Data-Intensive Systems. His research interests are energy-efficient memory oriented architectures including transactional memory, resilience, 3DIC, and networks on chip.

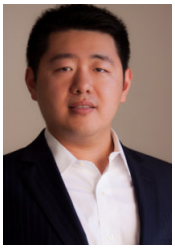


Bo Yuan received his Ph.D. degree from Department of Electrical and Computer Engineering at University of Minnesota, Twin cities in 2015, under the supervision of Prof. Keshab K. Parhi. He received the B.S. degree in physics and the M.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2007 and 2010, respectively. He is currently an assistant professor in the Department of Electrical Engineering at City University of New York (CUNY), City College (CCNY). He is also the affiliated faculty of the Computer Science Ph.D. Program in the CUNY Graduate Center. He is current interested in co-designing the algorithms (especially on artificial intelligence, machine learning and signal processing) and low-power fault-tolerant hardware to address the emerging challenges for embedded and intelligent systems in big data and IoT eras.



Jian Tang is a professor in the Department of Electrical Engineering and Computer Science at Syracuse University. He received his Ph.D degree in Computer Science from Arizona State University in 2006. His research interests lie in the areas of Cloud Computing, Big Data and Wireless Networking. Dr. Tang has published over 90 papers in premier journals and conferences. He received an NSF CAREER award in 2009, the 2016 Best Vehicular Electronics Paper Award from IEEE Vehicular Technology Society, and Best Paper Awards from the 2014 IEEE International Conference on Communications (ICC) and the 2015 IEEE Global Communications Conference (Globecom) respectively.

Qinru Qiu received her M.S. and Ph.D. degrees from the department of Electrical Engineering at University of Southern California in 1998 and 2001 respectively. She received her B.S. degree from the department of Information Science and Electronic Engineering at Zhejiang University, China in 1994. Dr. Qiu is currently a professor and the program director of Computer Engineering at Department of Electrical Engineering and Computer Science in Syracuse University. Before joining Syracuse University, she has been an assistant professor and then an associate professor at the Department of Electrical and Computer Engineering in State University of New York, Binghamton. Her research areas are high performance energy efficient computing systems, and neuromorphic computing. She is the TPC member of DATE, DAC, ISLPED, ISQED, VLSI-SoC, and ICCAD. She is the associate editor of TODAES.



Yanzhi Wang received the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2014. He is currently an assistant professor at the Department of Electrical Engineering and Computer Science at Syracuse University, Syracuse, NY, USA. His current research interests include system-level power management, neuromorphic computing, near-threshold computing, digital circuits power minimization and timing analysis, etc.