Towards Acceleration of Deep Convolutional Neural Networks using Stochastic Computing

Ji Li[1], Ao Ren[2], Zhe Li[2], Caiwen Ding[2], Bo Yuan[3], Qinru Qiu[2] and Yanzhi Wang[2]

[1]Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
[2]College of Engineering and Computer Science, Syracuse University, Syracuse, NY, USA
[3]Department of Electrical Engineering, City University of New York, NY, USA
jli724@usc.edu, {aren,zli89,cading}@syr.edu, byuan@ccny.cuny.edu, {qiqiu,ywang393}@syr.edu

Abstract—**In recent years, Deep Convolutional Neural Network (DCNN) has become the dominant approach for almost all recognition and detection tasks and outperformed humans on certain tasks. Nevertheless, the high power consumptions and complex topologies have hindered the widespread deployment of DCNNs, particularly in wearable devices and embedded systems with limited area and power budget. This paper presents a fully parallel and scalable hardware-based DCNN design using Stochastic Computing (SC), which leverages the energy-accuracy trade-off through optimizing SC components in different layers. We first conduct a detailed investigation of the Approximate Parallel Counter (APC) based neuron and multiplexer-based neuron using SC, and analyze the impacts of various design parameters, such as bit stream length and input number, on the energy/power/area/accuracy of the neuron cell. Then, from an architecture perspective, the influence of inaccuracy of neurons in different layers on the overall DCNN accuracy (i.e., software accuracy of the entire DCNN) is studied. Accordingly, a structure optimization method is proposed for a general DCNN architecture, in which neurons in different layers are implemented with optimized SC components, so as to reduce the area, power, and energy of the DCNN while maintaining the overall network performance in terms of accuracy. Experimental results show that the proposed approach can find a satisfactory DCNN configuration, which achieves 55X, 151X, and 2X improvement in terms of area, power and energy, respectively, while the error is increased by 2.86%, compared with the conventional binary ASIC implementation.**

I. Introduction

Machine learning technology is increasingly present in the Internet and consumer products, and it powers many fundamental applications such as speech-to-text transcription, selection of relevant search results, natural language processing, and objects identification in images or videos [1], [2]. Conventional machine learning techniques are limited in its ability to process data in their raw form (e.g., the pixel values of an image) [2]. As a result, considerable human engineering efforts and domain expertise are required to transform raw data into suitable internal representations that can be understood and processed by the learning system [2]. With the fast growing amount of data and range of applications to machine learning methods, the ability to automatically extract powerful features is becoming increasingly important [1].

Many representation learning methods have been proposed to automatically learn and organize the discriminative information from raw data [2]. *Deep learning* is one of the most promising representation learning methods, which enables a system to extract representations automatically at multiple levels of abstraction and learn complex functions directly from data with very little engineering by hand [1], [2]. With the self-learning ability to configure its intricate structure, a deep learning architecture can easily take advantage of increases in the amount of available computation and data [2].

Recently, Deep Convolutional Neural Network (DCNN), which is one of most widely used types of deep neural networks, has achieved tremendous success in many machine learning applications, such as speech recognition [3], image classification [4], and video classification [5]. DCNN is now the dominant approach for almost all recognition and detection tasks and approach human performance on some tasks [2]. Nevertheless, compared with other machine learning techniques, DCNNs require more computations due to the deep layer architecture. Furthermore, the industrial and academic demands for better quality of results also tend to increase the depth and/or width of DCNNs [6], leading to complicated topologies and increased computation resources required for implementation. Therefore, a practical implementation of large-scale DCNNs is to use high performance server clusters with accelerators such as GPUs and FPGAs [7], [8]. A notable trend is that with the astonishing advances on wearable devices and Internet-of-Things (IoT), machine learning has also been rapidly adopted in the widespread mobile and embedded systems. In order to bring the success of DCNNs to these resource constrained systems, designers must overcome the challenges of implementing resource-hungry DCNNs in embedded systems with limited area and power budget.

Stochastic Computing (SC), which is the paradigm of logical computation on stochastic bit streams [9], has the potential to enable fully parallel and scalable hardware-based DCNNs. Since SC provides several key advantages compared to conventional binary arithmetic, including low hardware area cost and tolerance to soft errors [9–11], considerable research efforts have been invested in the context of designing neural networks using SC in recent years [12–16].

Nevertheless, there lacks a comprehensive investigation of energy-accuracy trade-off for DCNN designs using different SC components. In this paper, two hardware-based neuron structures using SC are introduced, i.e., Accumulative Parallel Counter (APC) based neuron and multiplexer (MUX) based neuron. We further investigate the trade-off among area, power, energy and (neuron cell) accuracy for these neuron structures using different input sizes and stochastic bit stream lengths. Then, from an architecture perspective, the influence of inaccuracy of neurons in different layers on the overall DCNN accuracy is studied. Based on the results, a structure optimization method is proposed for a general DCNN architecture, in which neurons in different layers are implemented with the optimized SC components such that the overall DCNN area, power, and energy consumption are minimized while the DCNN accuracy is preserved.

The contributions of this work are threefold. First, we introduce SC into the DCNNs, in order to make the footprints of DCNNs small enough for successful implementations in today's wearable devices and embedded systems. Second, we carry out a detailed analysis on the energy-accuracy trade-off for different SC-based neuron designs. Third, based on the analysis of the results, we propose a structure optimization method for a general DCNN architecture using SC, which jointly optimizes the area, power, energy, and accuracy for the entire DCNN. Experimental results on a LeNet 5 DCNN architecture demonstrate that compared with the conventional 8-bit binary implementation, the presented hardware-based DCNN using SC achieves 55X, 151X, and 2X improvement in terms of area, power and energy, respectively, while the error is increased by 2.86%.

II. Related Work

DCNNs have been recognized as one of the most effective pattern recognition techniques. In order to improve the area, power and energy performance, many hardware-based neural networks have come into existence. The authors in [7] proposed an FPGA-based accelerator to leverage the sources of parallelism. An efficient 3D neuron topology was developed in [8], which improved the utilization of FPGA resources for different convolutional layer shapes.

In addition to accelerating techniques, SC becomes a very attractive candidate for implementing hardware-based neural networks, since SC building blocks can greatly reduce the hardware footprints, compared to conventional binary arithmetic components [17]. The authors in [14] applied SC to a radial basis function artificial neural network and significantly reduced the required hardware. Reference work [12] presented a neuron cell design using SC components, where the progressive precision characteristics of SC was exploited. Reconfigurable SC based neurons were developed in [15]. In addition, the authors in [16] explored the hardware-oriented pooling in DCNNs using SC. The above-mentioned works have proposed several neural network designs using SC, in order to satisfy the resource constraints in embedded systems whiling meeting the specific functions and performance needs of end users. However, there lacks a detailed investigation of the energy-accuracy trade-offs for DCNNs using different SC components. Moreover, within a DCNN architecture, neurons in different layers have various connection patterns and exhibit different degrees of influence on the overall system performance, which indicates a structure optimization can be applied to achieve further improvement.

III. Overview of the Proposed DCNN and Stochastic Computing

A. DCNN Architecture

In this paper, we consider a general DCNN architecture, which consists of a stack of convolutional layers, pooling layers, and fully connected layers. By arranging the topology of above layers, powerful architectures (e.g., LeNet [18]) can be built for specific applications. Without the loss of generality, we conduct the investigation on the LeNet-5 architecture using SC, which is comprised of two pairs of convolutional and pooling layers, one fully connected layer, and one output layer, as shown in Figure 1. Note that the proposed methodology can accommodate other DCNN architectures as well.

A convolutional layer is associated with a set of learnable filters (or kernels), which are activated when specific types of features are found at some spatial positions in the inputs. After obtaining features using convolution, a subsampling step can be applied to aggregate statistics of these features to reduce the dimensions of data and mitigate over-fitting issues. This subsampling operation is realized by a pooling layer in hardware-based DCNNs, where different non-linear functions can be applied, such as max pooling, average pooling and L2-norm pooling. The activation functions in neurons 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 $f(x) = \frac{1}{1+e^{-x}}$. In this paper, we adopt the tanh activation function since it can be implemented efficiently as a finite state machine (FSM) in SC using a stochastic approximation method. Fully connected layer is a normal neural network layer with its inputs fully connected with its previous layer. The loss function of DCNN that specifies how the network training penalizes the deviation between the predicted and true labels, and typical loss functions are softmax loss, sigmoid cross-entropy loss or Euclidean loss.

B. Stochastic Computing

In SC, the value of a (unipolar) stochastic number is represented by the probability of 1s in a random bit stream, e.g., the value of a 4-bit sequence $X = 0010$ is $x = P(X = 1) = \frac{1}{4} = 0.25$. Obviously, the representation of a stochastic number is not unique, e.g., to represent the value 0.25 using a 4-bit stream, there are four different ways: 0001, 0010, 0100, and 1000. Besides, an m-bit sequence can only represent numbers in the set $\{\frac{0}{m}, \frac{1}{m}, \frac{2}{m}, \cdots, \frac{m}{m}\}$, indicating that only a small subset of the real numbers in the interval $[0, 1]$ can be expressed exactly in SC. Clearly, the precision and accuracy of SC is dependent on the length of the stream. The two most popular representations for stochastic numbers are unipolar and bipolar formats, which interpret values in the intervals $[0, 1]$ and $[-1, 1]$, respectively. Unipolar coding is commonly used in unsigned arithmetic operations, whereas bipolar format is used in signed arithmetic calculations. More specifically, in unipolar coding, the information carried in a stochastic stream of bits X is $x = P(X = 1) = P(X)$, whereas in the bipolar format, $x = 2P(X = 1) - 1 = 2P(X) - 1$.

The major arithmetic operations included in DCNN are multiplication, addition, and tanh. With SC, these operations can be implemented with extremely small circuits as follows.

1) Multiplication: Stochastic multiplication in unipolar and bipolar is performed by an AND gate and an XNOR gate, respectively. We denote the probabilities of 1 on the input bit streams by $P(A)$ and $P(B)$, and the probability of 1 at the output of the AND gate is $P(A) \times P(B)$, i.e., the product of unipolar multiplication. As for the bipolar coding, the output

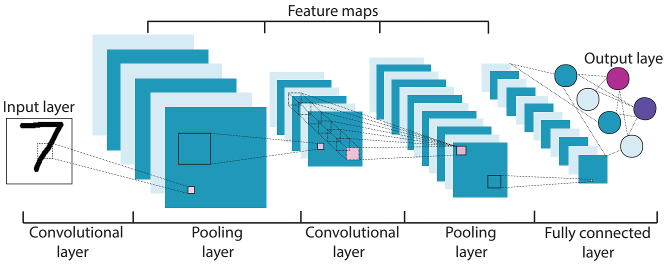

Fig. 1. The fifth generation of LeNet DCNN architecture.

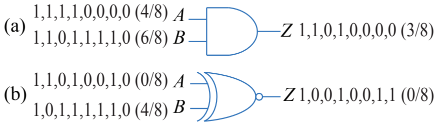

Fig. 2. Stochastic multiplication: (a) unipolar and (b) bipolar.

of the XNOR gate is $P(Z) = P(A) \cdot P(B) + \overline{P}(A) \cdot \overline{P}(B)$. Therefore, the stochastic number of Z is calculated as $z = 2 \cdot P(Z) - 1 = 4 \cdot P(A) \cdot P(B) - 2 \cdot P(A) - 2 \cdot P(B) + 1 = (2 \cdot P(A) - 1) \cdot (2 \cdot P(B) - 1) = a \times b$. Note that the input bit streams are assumed to be suitably uncorrelated or independent in the above calculations. An example of multiplying two bit streams A and B is illustrated in Figure 2.

2) Addition: Stochastic addition can be implemented by an OR gate, a multiplexer (MUX), or an Accumulative Parallel Counter (APC) [19], as illustrated in Figure 3 (a), (b), and (c), respectively. When both inputs a and b are small, the output of the OR gate is an approximate sum that is expressed as $z = P(Z) = P(A) + P(B) - P(A \cdot B) \approx a + b$. A MUX performs scaled addition by randomly selecting one input i among n inputs with probability p_i such that $\sum_{i=1}^{n} p_i = 1$. For example, adding a and b using MUX with $p_1 = p_2 = 50\%$ generates an output $z = P(Z) = \frac{1}{2} \cdot \big(P(A) + P(B)\big)$. This MUX can also perform scaled addition in bipolar coding, i.e., $z = 2 \cdot P(Z) - 1 = \frac{1}{2} \cdot \big((2 \cdot P(A) - 1) + (2 \cdot P(B) - 1)\big) = \frac{1}{2} \cdot (a + b)$. However, the MUX has the drawback of losing $n - 1$ inputs information, since only one bit is selected and the remaining $n - 1$ bits are ignored at a time. In order to achieve better accuracy, APC is proposed to compute the total number of 1s present in all the inputs using a parallel counter. Note that the output of APC is in binary, so additional steps may be needed to transform this. In conclusion, OR gate is the most area efficient but the accuracy is too low to be used in DCNN. MUX is area efficient with limited accuracy, whereas APC achieves better accuracy at the cost of a larger footprint.

3) Hyperbolic Tangent: The hyperbolic tangent function (i.e., $tanh(\cdot)$) is implemented using a K-state FSM, as shown in Figure 4, where half of the states generate output 0 and the other half states generate 1. According to [20], for a given bipolar stochastic number x, the result of the FSM design is a stochastic approximation to the $tanh$ function as follows,

$$Stanh(K, x) = tanh(\frac{K \cdot x}{2}) \tag{1}$$

Therefore, additional conversion steps are needed to calculate the $tanh(x)$. The accuracy of the K-state FSM $tanh$ function is determined by state number K and input stream length.

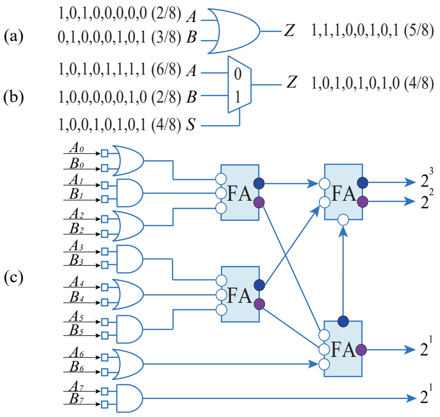

Fig. 3. Stochastic adders: (a) OR gate, (b) multiplexer (MUX) for scaled addition, and (c) approximate parallel counters (APC).

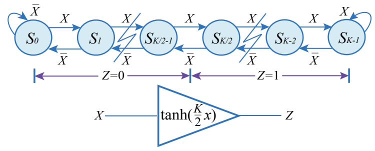

Fig. 4. Stochastic hyperbolic tangent.

IV. Hardware-Based DCNN Design and Optimization using Stochastic Circuits

In this section, we first conduct a detailed investigation of the energy-accuracy trade-off among two hardware neuron designs using SC, i.e., APC-based neuron and MUX-based neuron, as shown in Figure 5 (a) and (b), respectively. Hardware-based pooling is provided afterward, and finally we present the structure optimization method for the overall DCNN architecture.

A. APC-Based Neuron

Figure 5 (a) illustrates the APC-based hardware neuron design, where the inner product is calculated using XNOR gates (for multiplication) and an APC (for addition). To be more specific, we denote the number of bipolar inputs and stochastic stream length by n and m, respectively. Accordingly, n XNOR gates are used to generate n products of inputs ($x_i's$) and weights ($w_i's$), and then the APC accumulates the sum of 1s in each column of the products. Since the sum generated by APC is a binary number, the K-state FSM design mentioned in Section III-B3 cannot be applied here directly. Instead of an FSM, a saturated up/down counter is used to perform the scaled hyperbolic tangent activation function $Btanh(\cdot)$ for binary inputs. Details and optimization of the $Btanh(\cdot)$ activation function using a saturated up/down counter for binary inputs can be found in reference work [12].

For an APC-based neuron with the fixed bit stream length 1024, the accuracy, area, power, and energy performance with respect to the input size are shown in Figure 6 (a), (b), (c), and (d), respectively. To be more specific, as illustrated in Figure 6 (a), APC-based neuron shows a very slow accuracy degradation as input size increases. However, the area, power, and energy of the entire APC-based neuron cell increases near linearly as the input size grows, as shown in Figure 6 (b), (c), and (d), respectively. The reason is as follows: With the efficient implementation of $Btanh(\cdot)$ function, the hardware

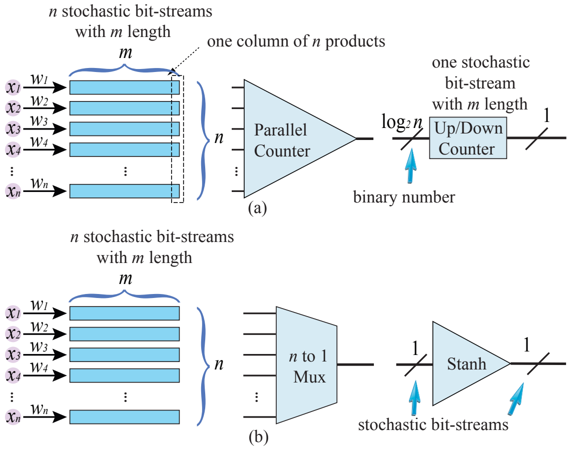

Fig. 5. Various hardware neuron designs. (a) APC-based neuron, and (b) MUX-based neuron.

of $Btanh(\cdot)$ increases logarithmically as the input increases, since the input width of $Btanh(\cdot)$ is $log_2 n$. On the other hand, the number of XNOR gates and the size of the APC grow linearly as the input size increases. Hence, the inner product calculation part, i.e., XNOR array and APC, is dominant in an APC-based neuron, and the area, power, and energy of the entire APC-based neuron cell also increase at the same rate as the inner product part when the input size increases.

Since the length of the stochastic bit stream is important, we investigate the accuracy of APC-based neurons using different stream lengths under different input sizes. As shown in Figure 7, longer bit stream length consistently outperforms lower bit stream length in terms of accuracy in APC-based neurons with different input sizes. However, designers should consider the latency and energy overhead caused by long bit streams.

B. MUX-Based Neuron

As shown in Figure 5 (b), a MUX-based neuron is comprised of XNOR gates, a MUX, and a K-state FSM, in order to compute the products of bipolar inputs ($x_i' s$) and weights ($w_i' s$), the stochastic sum of all products, and the hyperbolic tangent activation function, respectively. As the inner product calculated by a MUX is a stochastic number, the K-state FSM design mentioned in Section III-B3 can be used here to implement the activation function $Stanh(\cdot)$.

Nevertheless, two problems must be taken into consideration: (i) the inner product calculated by an n input MUX is scaled to $\frac{z}{n}$, assuming the correct result is z, and (ii) with the input $\frac{z}{n}$, the K-state FSM calculates $tanh(\frac{K\cdot z}{2\cdot n})$ instead of the desired value $tanh(z)$. Hence, in order to get the correct activation, we need to scale up the results of MUX by n times and multiply the stream by $\frac{2}{K}$ (or multiply by $\frac{2\cdot n}{K}$ directly). As opposed to the relatively simple and efficient data conversions on a software platform, such conversions in a hardware-based neuron incurs significant hardware overhead, because the linear gain transformation needs one more FSM [20], and the multiplication requires one XNOR gate as well as the generation of the other bipolar stochastic stream.

In this paper, considering an n inputs neuron with inner product denoted by z, we select the state number K such that $\frac{2\cdot n}{K} = 1$, and the final output of the FSM is calculated as

$$Stanh(K, \frac{z}{n}) = tanh(\frac{K\cdot z}{2\cdot n}) = tanh(z) \tag{2}$$

In this way, we achieve the desired activation result with no additional bit stream conversion (i.e., no hardware overhead).

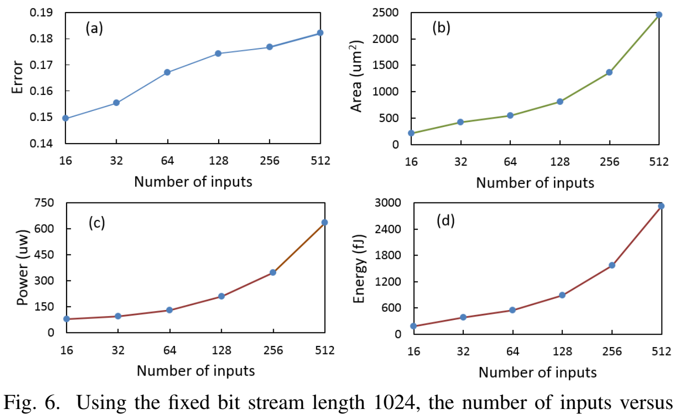

Fig. 6. Using the fixed bit stream length 1024, the number of inputs versus (a) accuracy, (b) area, (c) power and (d) energy for an APC-based neuron.

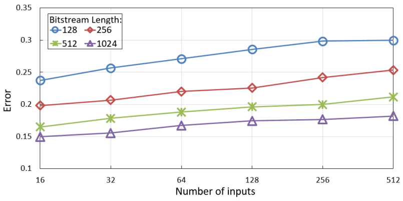

Fig. 7. The length of bit stream versus accuracy under different input numbers for an APC-based neuron.

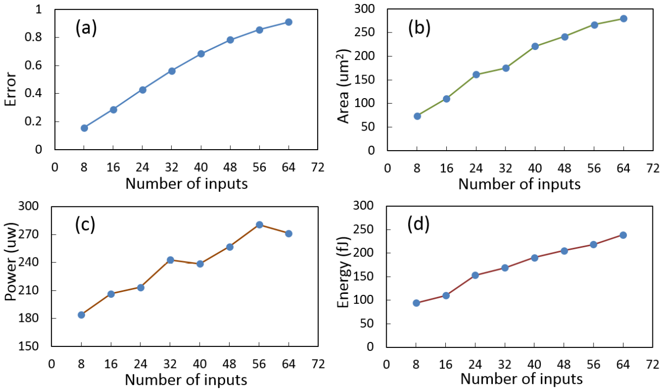

Fig. 8. Using the fixed bit stream length 1024, the number of inputs versus (a) accuracy, (b) area, (c) power and (d) energy for a MUX-based neuron.

We first investigate the performance of the MUX-based neuron with respect to its input size. Figure 8 (a), (b), (c), and (d) show the results of the number of inputs versus accuracy, area, power, and energy, respectively, for a MUX-based neuron using a fixed bit stream length which is equal to 1024. It is important to achieve a high accuracy of a neuron cell, however, as shown in Figure 8 (a), the accuracy of a MUX-based neuron significantly degrades as the input size increases. The reason is that MUX addition selects only one bit at a time and ignores the rest of the bits, leading to low accuracy when input size is large. In addition, one can observe from Figure 8 (b), (c), and (d) that as the number of inputs increases, area, power, and energy of the MUX-based neuron all tend to increase. This is because a MUX-based neuron with more inputs requires more XNOR gates and MUXes for inner product calculation, and more states in the FSM ($K = 2 \cdot n$) to compute the activation function. Hence, the increased hardware components result in more area, power, and energy of the neuron cell.

Next, we investigate the relationship between bit stream length and accuracy under different numbers of inputs. As shown in Figure 9, for a certain input size, longer bit stream results in higher accuracy, and the improvement of accuracy is more significant when input size is larger. Hence, when designing a MUX-based neuron, long bit stream can be applied to compensate the accuracy degradation for large input size.

C. Pooling Operation

In a DCNN, down sampling steps are performed by the pooling layers, which summarize the outputs of neighboring groups of neurons in the same kernel map. Pooling operation achieves the invariance to input data (i.e., image, video, etc.) transformations and better robustness to noise and clutter. Moreover, the inter-layer connections can be significantly reduced for a hardware DCNN by using pooling layers.

Considering a pooling region consisting of k neurons: $\{a_1, \cdots, a_k\}$ in a feature map, where a_i denotes the activation result of the i-th neuron, the pooling layer selects one activation a_{out} at a time. In this paper, we adopt the *average pooling*,

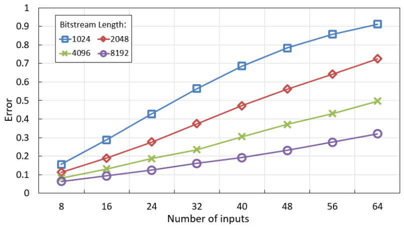

Fig. 9. The length of bit stream versus accuracy under different input numbers for a MUX-based neuron.

where each activation result a_i has the same probability to be selected as output, i.e., $a_{out} = mean(a_1 \sim a_k)$. For example, the stochastic arithmetic mean over a 2×2 region is provided in Figure 10, where three 2-to-1 MUXes are needed to implement the average pooling.

D. Structure Optimization for the Entire DCNN Architecture

There are four performance metrics for the DCNN design, i.e., accuracy, area, power, and energy. In this paper, we consider a general DCNN optimization problem, where the objective function is comprised of one or multiple metrics and the rest of the metrics are considered as constraints, e.g., energy, power, and accuracy as objective function with area as constraint. In addition, we introduce one more constraint that the accuracy of hardware-based DCNN cannot be significantly lower than the accuracy of software-based DCNN, so as to make the accuracy of the hardware-based DCNN competitive.

The DCNN architecture of interest shown in Figure 1 consists of two pooling layers, two convolutional layers, and one fully-connected layer. The two pooling layers are implemented using MUX trees, as described in Section IV-C. As for the remaining two convolutional layers (referred to as layer 0 and layer 1) and one fully-connected layer (referred to as layer 2), they can be built using either APC-based neurons or MUX-based neurons with a certain bit stream length.

We further investigate the influences of errors in layer 0, layer 1 and layer 2 on the overall test error of the entire DCNN, as shown in Figure 11, where the data values in each layer follow a normal distribution (as observed in the test benches) with various standard deviations representing the errors of the neurons in that layer. It is observed that a layer closer to the inputs has more impact on the overall accuracy of the DCNN than a layer closer to the output layer. The explanation is that inaccurate features captured near the inputs may affect all the following layers, whereas the errors occurring near the output layer can only disturb a few subsequent layers. Therefore, the intuition is that accurate neuron structures should be applied to the layers near inputs, and less accurate neurons can be used in the layers closer to the output layer to achieve better energy/power/area performance.

Next, we compare the performance between APC-based neuron and MUX-based neuron using a fixed bit stream length equal to 1024 under different input sizes, as shown in Table I. Clearly, APC-based neuron is more accurate but occupies more area than MUX-based neuron. Besides, as APC is much slower than MUX, the latency of APC-based neuron is larger than MUX-based neuron, which causes APC-based neuron to consume more energy than MUX-based neuron for one calculation. As for the power performance, an APC-based neuron has less switching (due to the long latency) and larger area than the MUX-based neuron, resulting in less dynamic power, more leakage power, and less overall power.

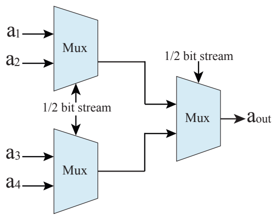

Fig. 10. A 4-to-1 pooling example.

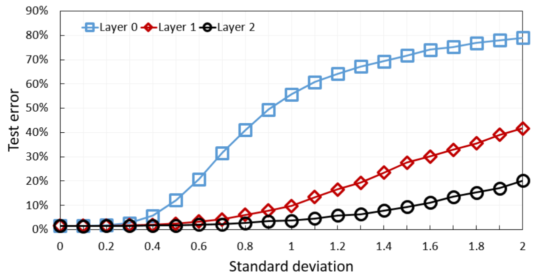

Fig. 11. The impact of errors in different layers on the overall DCNN test error.

TABLE I
COMPARISON BETWEEN APC-BASED NEURON AND MUX-BASED NEURON USING 1024 BIT STREAM

	APC-based neuron			MUX-based neuron			Ratio of APC/MUX (%)		
Input size	16	32	64	16	32	64	16	32	64
Absolute error	0.15	0.16	0.17	0.29	0.56	0.91	51.94	27.56	18.34
Area (μm^2)	209.9	417.6	543.2	110.7	175.3	279.8	189.7	238.2	194.1
Power (μW)	80.7	95.9	130.5	206.5	242.9	271.2	39.1	39.5	48.1
Energy (fJ)	177.4	383.7	548.1	110.0	169.1	238.9	161.3	226.9	229.5

The proposed structure optimization method for the overall DCNN architecture is given in Figure 12. As the bit stream length significantly affects the energy consumption and accuracy of the entire DCNN, the first step is to apply binary search to choose a suitable bit stream length for a DCNN configuration (i.e., neuron structure configuration in each layer). Note that the DCNN configuration used in step 1 is not important as the results will be refined in the following steps. In step 2, under the fixed bit stream length, all the promising configurations are explored, where some configurations can be ruled out, e.g., all layers using MUX-base neurons is highly inaccurate and can be ruled out. Based on the results of step 2, the configurations with desirable performance will be selected, and in the following step 3, for each configuration, we try other bit stream lengths to see if better performance can be achieved. The final configuration of the DCNN is decided based on the result of step 3, and several more iterations may be needed to further refine the result by exploring more configurations.

V. EXPERIMENTAL RESULTS

The LeNet5 DCNN used in this experiment is built with a $784-11520-2880-3200-800-500-10$ configuration. The MNIST handwritten digit image dataset [21] consists of 60,000 training data and 10,000 testing data with 28x28 grayscale image and 10 classes is used in the experiments, and the network is trained with 20 epochs (batch size =500). We use Synopsys Design Compiler to synthesize the DCNNs with the 45nm Nangate Open Cell Library [22].

Table II concludes the configurations and performance for all the explored hardware-based DCNNs (No. $1-15$) using the proposed structure optimization method, the 8 bit conventional binary pipelined baseline (No. 16) and software-based DCNNs using CPU (No. 17) or GPU (No. 18) for comparison. Note that the power for software is estimated using Thermal Design Power (TDP), and the energy is calculated by multiplying the run time and TDP.

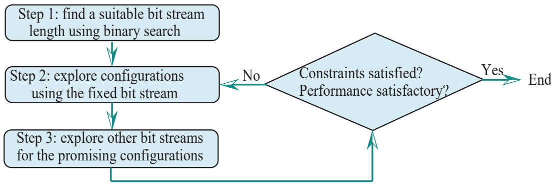
Step 1: find a suitable bit stream length using binary search
Step 2: explore configurations using the fixed bit stream
No
Constraints satisfied? Performance satisfactory?
Yes
End
Step 3: explore other bit streams for the promising configurations

Fig. 12. Structure optimization method for the entire DCNN.

Without any loss of generality, we set the desired accuracy to be $\leq 4.5\%$ error rate. In the first step of the proposed structure optimization method, the bit stream length is set to 1024 using binary search. In step 2, using the fixed bit stream length, all configurations are explored, as shown in Table II (No. $1-8$). DCNNs in No. $1-5$ are ruled out due to the low accuracy, and in step 3, the remaining promising DCNNs in No. $6-8$ are explored using the decreased bit stream length 512 bits, where the results are given as DCNNs in No. $9-11$. Since DCNNs No. $10-11$ satisfy the accuracy constraint, we further reduce the bit stream to 256 to improve energy performance, where DCNNs in No. $12-13$ provide the results. This time, only DCNNs in No. 13 (all APC-based neurons) meet the accuracy constraint ($\leq 4.5\%$). Hence, the bit stream length is further reduced for DCNN in No. 13 so as to find the configuration that achieves the minimum energy while satisfying the accuracy constraint.

The DCNNs that use more MUX-based neurons provide smaller footprints, which are suitable for area-constraint embedded systems, whereas the DCNNs with more APC-based neurons achieve better accuracy, energy and power, which are good for power/energy-constraint embedded systems. Given the constraint(s), the proposed structure optimization method can provide the DCNN configurations with satisfactory performance. For instance, DCNNs in No. $10, 11, 13-15$ are all promising configurations found by the proposed method, given the accuracy constraint. Compared with the conventional 8-bit binary implementation, the presented hardware-based DCNN using SC (No. 15) achieves 55X, 151X, and 2X improvement in terms of area, power and energy, respectively, while the error is increased by 2.86%.

TABLE II
COMPARISON AMONG VARIOUS HARDWARE-BASED DCNNS AND SOFTWARE-BASED DCNNS

No.	Bit Stream	Layer 0, 1, 2	Error (%)	Area (mm^2)	Power (W)	Energy (μJ)
1	1024	MUX, MUX, MUX	21.66	6.62	3.3	4.4
2	1024	MUX, MUX, APC	11.89	7.42	1.3	6.7
3	1024	MUX, APC, MUX	16.25	8.05	1.5	6.9
4	1024	MUX, APC, APC	8.68	8.85	1.7	8.7
5	1024	APC, MUX, MUX	7.69	11.75	2.6	12.0
6	1024	APC, MUX, APC	2.49	12.56	2.7	13.8
7	1024	APC, APC, MUX	4.32	13.18	3.0	14.0
8	1024	APC, APC, APC	1.70	13.98	3.1	15.8
9	512	APC, MUX, APC	4.66	12.56	2.7	6.9
10	512	APC, APC, MUX	4.45	13.18	3.0	7.0
11	512	APC, APC, APC	1.70	13.98	3.1	7.9
12	256	APC, APC, MUX	5.20	13.18	3.0	3.5
13	256	APC, APC, APC	2.00	13.98	3.1	4.0
14	128	APC, APC, APC	2.34	13.98	3.1	2.0
15	64	APC, APC, APC	4.40	13.98	3.1	1.0
16	8 bit fixed point binary(pipelined)		1.54	769.30	470.0	2.0
17	CPU: two Intel Xeon W5580		1.54	263	130.0	198200
18	GPU: NVIDIA Tesla C2075		1.54	520	225.0	96443

VI. CONCLUSION

In this paper, two hardware-based neuron structures using SC were analyzed, and the influence of inaccuracy of neurons in different layers on the overall DCNN accuracy was studied. A structure optimization method was proposed for a general DCNN architecture, which jointly optimized the accuracy, area, power, and energy. Experimental results demonstrated that compared with the binary ASIC DCNNs, the area, power and energy of the hardware-based DCNN generated by the proposed structure optimization were significantly improved, whereas the accuracy performance was slightly degraded.

REFERENCES

[1] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8614–8618.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[5] A. Karpathy *et al.*, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[6] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[7] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of fpga-based deep convolutional neural networks," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 575–580.

[8] A. Rahman, J. Lee, and K. Choi, "Efficient fpga acceleration of convolutional neural networks using logical-3d compute array," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1393–1398.

[9] P. Li *et al.*, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2012, pp. 480–487.

[10] J. Li and J. Draper, "Accelerating soft-error-rate (ser) estimation in the presence of single event transients," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 55.

[11] J. Li and J. Draper, "Joint soft-error-rate (ser) estimation for combinational logic and sequential elements," in *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE, 2016, pp. 737–742.

[12] K. Kim *et al.*, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 124.

[13] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou, "Fpga implementation of a deep belief network architecture for character recognition using stochastic computation," in *Information Sciences and Systems (CISS), 2015 49th Annual Conference on*. IEEE, 2015, pp. 1–5.

[14] Y. Ji, F. Ran, C. Ma, and D. 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*. EDA Consortium, 2015, pp. 880–883.

[15] A. Ren *et al.*, "Designing reconfigurable large-scale deep learning systems using stochastic computing," in *2016 IEEE International Conference on Rebooting Computing. IEEE*, 2016.

[16] Z. Li *et al.*, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016.

[17] B. D. Brown and H. C. Card, "Stochastic neural computation. ii. soft competitive learning," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 906–920, 2001.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] P.-S. Ting and J. P. Hayes, "Stochastic logic realization of matrix operations," in *Digital System Design (DSD), 2014 17th Euromicro Conference on*. IEEE, 2014, pp. 356–364.

[20] B. D. Brown and H. C. Card, "Stochastic neural computation. i. computational elements," *IEEE Transactions on computers*, vol. 50, no. 9, pp. 891–905, 2001.

[21] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[22] Nangate 45nm Open Library, Nangate Inc., 2009. [Online]. Available: http://www.nangate.com/