SC-DCNN: Highly-Scalable Deep Convolutional Neural Network using Stochastic Computing

Ao Ren[†] Ji Li[◇] Zhe Li[†] Caiwen Ding[†]
Xuehai Qian[◇] Qinru Qiu[†] Bo Yuan[∓] Yanzhi Wang[†]
[†] *Department of Electrical Engineering and Computer Science, Syracuse University*
[◇] *Department of Electrical Engineering, University of Southern California*
[∓] *Department of Electrical Engineering, City University of New York, City College*
[†] {*aren, zli89, cading, qiqiu, ywang393*}*@syr.edu,*
[◇] {*jli724, xuehai.qian*}*@usc.edu,* [∓] *byuan@ccny.cuny.edu*

Abstract

With the recent advance of wearable devices and Internet of Things (IoTs), it becomes attractive to implement the Deep Convolutional Neural Networks (DCNNs) in embedded and portable systems. Currently, executing the software-based DCNNs requires high-performance servers, restricting the widespread deployment on embedded and mobile IoT devices. To overcome this obstacle, considerable research efforts have been made to develop highly-parallel and specialized DCNN accelerators using GPGPUs, FPGAs or ASICs.

Stochastic Computing (SC), which uses a bit-stream to represent a number within [-1, 1] by counting the number of ones in the bit-stream, has high potential for implementing DCNNs with high scalability and ultra-low hardware footprint. Since multiplications and additions can be calculated using AND gates and multiplexers in SC, significant reductions in power (energy) and hardware footprint can be achieved compared to the conventional binary arithmetic implementations. The tremendous savings in power (energy) and hardware resources allow immense design space for enhancing scalability and robustness for hardware DCNNs.

This paper presents SC-DCNN, the first comprehensive design and optimization framework of SC-based DCNNs, using a bottom-up approach. We first present the designs of function blocks that perform the basic operations in DCNN, including inner product, pooling, and activation function. Then we propose four designs of feature extraction blocks, which are in charge of extracting features from input feature maps, by connecting different basic function blocks with joint optimization. Moreover, the efficient weight storage methods are proposed to reduce the area and power (energy) consumption. Putting all together, with feature extraction blocks carefully selected, SC-DCNN is holistically optimized to minimize area and power (energy) consumption while maintaining high network accuracy. Experimental results demonstrate that the LeNet5 implemented in SC-DCNN consumes only 17 mm^2 area and 1.53 W power, achieves throughput of 781250 images/s, area efficiency of 45946 images/s/mm^2, and energy efficiency of 510734 images/J.

ASPLOS '17, April 08-12, 2017, Xi'an, China

DOI: http://dx.doi.org/10.1145/3037697.3037746

1. Introduction

Deep learning (or deep structured learning) has emerged as a new area of machine learning research, which enables a system to automatically learn complex information and extract representations at multiple levels of abstraction (10). *Deep Convolutional Neural Network (DCNN)* is recognized as one of the most promising types of artificial neural networks taking advantage of deep learning and has become the dominant approach for almost all recognition and detection tasks (27). Specifically, DCNN has achieved significant success in a wide range of machine learning applications, such as image classification (37), natural language processing (8), speech recognition (35), and video classification (19).

Currently, the high-performance servers are usually required for executing software-based DCNNs since software-based DCNN implementations involve a large amount of computations to achieve outstanding performance. However, the high-performance servers incur high power (energy) consumption and large hardware cost, making them unsuitable for applications in embedded and mobile IoT devices that require low-power consumption. These applications play an increasingly important role in our everyday life and exhibit a notable trend of being "smart". To enable DCNNs in these application with low-power and low-hardware cost, the highly-parallel and specialized hardware has been de-

signed using General-Purpose Graphics Processing Units (GPGPUs), Field-Programmable Gate Array (FPGAs), and Application-Specific Integrated Circuit (ASICs) (25; 38; 46; 31; 5; 41; 4; 40; 15; 44; 7; 14). Despite the performance and power (energy) efficiency achieved, a large margin of improvement still exists due to the inherent inefficiency in implementing DCNNs using conventional computing methods or using general-purpose computing devices (16; 21).

We consider *Stochastic Computing (SC)* as a novel computing paradigm to provide significantly low hardware footprint with high energy efficiency and scalability. In SC, a probability number is represented using a bit-stream (13), therefore, the key arithmetic operations such as multiplications and additions can be implemented as simple as AND gates and multiplexers (MUX), respectively (6). Due to these features, SC has the potential to implement DCNNs with significantly reduced hardware resources and high power (energy) efficiency. Considering the large number of multiplications and additions in DCNN, achieving the efficient DCNN implementation using SC requires the exploration of a large design space.

In this paper, we propose *SC-DCNN*, the first comprehensive design and optimization framework of SC-based DCNNs, using a bottom-up approach.

The proposed SC-DCNN fully utilizes the advantages of SC and achieves remarkably low hardware footprint, low power and energy consumption, while maintaining high network accuracy. Based on the proposed SC-DCNN architecture, this paper made the following key contributions:

- **Applying SC to DCNNs.** We are the *first* (to the best of our knowledge) to apply SC to DCNNs. This approach is motivated by *1)* the potential of SC as a computing paradigm to provide low hardware footprint with high energy efficiency and scalability; and *2)* the need to implement DCNNs in the embedded and mobile IoT devices.
- **Basic function blocks and hardware-oriented max pooling.** We propose the design of *function blocks* that perform the basic operations in DCNN, including Specifically, we present a novel hardware-oriented max pooling design for efficiently implementing (approximate) max pooling in SC domain. The pros and cons of different types of function blocks are also thoroughly investigated.
- **Joint optimizations for feature extraction blocks.** We propose four optimized designs of *feature extraction blocks* which are in charge of extracting features from input feature maps. The function blocks inside the feature extraction block are *jointly optimized* through both analysis and experiments with respect to input bit-stream length, function block structure, and function block compatibilities.
- **Weight storage schemes.** The area and power (energy) consumption of weight storage are reduced by comprehensive techniques, including efficient filter-aware SRAM sharing, effective weight storage methods, and layer-wise weight storage optimizations.
- **Overall SC-DCNN optimization.** We conduct holistic optimizations of the overall SC-DCNN architecture with carefully selected feature extraction blocks and layer-wise feature extraction block configurations, to minimize area and power (energy) consumption while maintaining the high network accuracy. The optimization procedure leverages the crucial observation that hardware inaccuracies in different layers in DCNN have different effects on the overall accuracy. Therefore, different designs may be exploited to minimize area and power (energy) consumptions.
- **Remarkably low hardware footprint and low power (energy) consumption.** Overall, the proposed SC-DCNN achieves the *lowest* hardware cost and energy consumption in implementing LeNet5 compared with reference works.

2. Related Works

Authors in (25; 38; 23; 17) leverage the parallel computing and storage resources in GPUs for efficient DCNN implementations. FPGA-based acceleration (46; 31) is another promising path towards the hardware implementation of DCNNs due to the programmable logic, high degree of parallelism and short develop round. However, the GPU and FPGA-based implementations still exhibit a large margin of performance enhancement and power reduction. It is because *1)* 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 restricts the performance of DCNNs which typically exhibit high inter-neuron communication requirements.

ASIC-based implementations of DCNNs have been recently exploited to overcome the limitations of general-purpose computing devices. Two representative recent works are DaDianNao (7) and EIE (14). The former proposes an ASIC "node" which could be connected in parallel to implement a large-scale DCNN, whereas the latter focuses specifically on the fully-connected layers of DCNN and achieves high throughput and energy efficiency.

To significantly reduce hardware cost and improve energy efficiency and scalability, novel computing paradigms need to be investigated. We consider SC-based implementation of neural network an attractive candidate to meet the stringent requirements and facilitate the widespread of DCNNs in embedded and mobile IoT devices. Although not focusing on deep learning, (36) proposes the design of a neurochip using stochastic logic. (16) utilizes stochastic logic to implement a radial basis function-based neural network. In addition, a neuron design with SC for deep belief network was presented in (21). Despite the previous application of SC, there

is no existing work that investigates comprehensive designs and optimizations of SC-based hardware DCNNs including both computation blocks and weight storing methods.

3. Overview of DCNN Architecture and Stochastic Computing

3.1 *DCNN Architecture Overview*

Deep convolutional neural networks are biologically inspired variants of multilayer perceptrons (MLPs) by mimicking the animal visual mechanism (2). An animal visual cortex contains two types of cells and they are only sensitive to a small region (receptive field) of the visual field. Thus a neuron in a DCNN is only connected to a small receptive field of its previous layer, rather than connected to all neurons of previous layer like traditional fully connected neural networks.

As shown in Figure 1, in the simplest case, a DCNN is a stack of three types of layers: *Convolutional Layer*, *Pooling Layer*, and *Fully Connected Layer*. The Convolutional layer is the core building block of DCNN, its main operation is the convolution that calculates the dot-product of receptive fields and a set of learnable filters (or kernels) (1). Figure 2 illustrates the process of convolution operations. After the convolution operations, the nonlinear down-samplings are conducted in the pooling layers to reduce the dimension of data. The most common pooling strategies are *max pooling* and *average pooling*. Max pooling picks up the maximum value from the candidates, and average pooling calculates the average value of the candidates. Then, the extracted feature maps after down-sampling operations are sent to *activation functions* that conduct non-linear transformations such as Rectified Linear Unit (ReLU) $f(x) = max(0, x)$, Sigmoid function $f(x) = (1 + e^{-x})^{-1}$ and hyperbolic tangent (tanh) function $f(x) = \frac{2}{1+e^{-2x}} - 1$. Next, the high-level reasoning is completed via the fully connected layer. Neurons in this layer are connected to all activation results in the previous layer. Finally, the loss layer is normally the last layer of DCNN and it specifies how the deviation between the predicted and true labels is penalized in the network training process. Various loss functions such as softmax loss, sigmoid cross-entropy loss may be used for different tasks.

The concept of "neuron" is widely used in the software/algorithm domain. In the context of DCNNs, a neuron consists of one or multiple basic operations. In this paper, we focus on the basic operations in hardware designs and optimizations, including: inner product, pooling, and

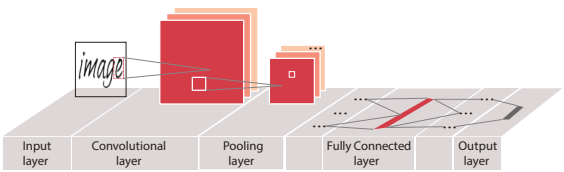

Figure 1. The general DCNN architecture.

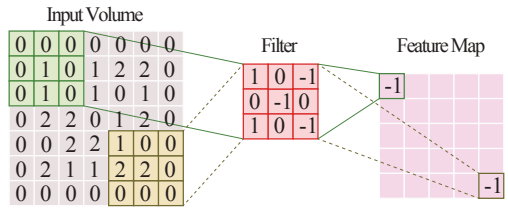

Figure 2. Illustration of the convolution process.

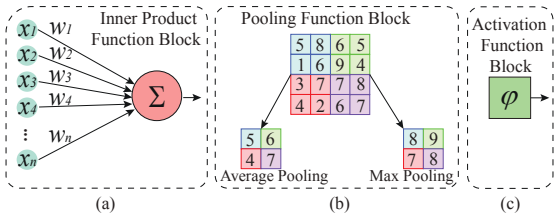

Figure 3. Three types of basic operations (function blocks) in DCNN. (a) Inner Product, (b) pooling, and (c) activation.

activation. The corresponding SC-based designs of these fundamental operations are termed *function blocks*. Figure 3 illustrates the behaviors of function blocks, where x_i's in Figure 3(a) represent the elements in a receptive filed, and w_i's represent the elements in a filter. Figure 3(b) shows the average pooling and max pooling function blocks. Figure 3(c) shows the activation function block (e.g. hyperbolic tangent function). The composition of an inner product block, a pooling block, and an activation function block is referred to as the *feature extraction block*, which extracts features from feature maps.

3.2 Stochastic Computing (SC)

Stochastic Computing (SC) is a paradigm that represents a probabilistic number by counting the number of ones in a bit-stream. For instance, the bit-stream 0100110100 contains four ones in a ten-bit stream, thus it represents $P(X = 1) = 4/10 = 0.4$. In addition to this unipolar encoding format, SC can also represent numbers in the range of [-1, 1] using the bipolar encoding format. In this scenario, a real number x is processed by $P(X = 1) = (x + 1)/2$, thus 0.4 can be represented by 1011011101, as $P(X = 1) = (0.4 + 1)/2 = 7/10$. To represent a number beyond the range [0, 1] using unipolar format or beyond [-1, 1] using bipolar format, a pre-scaling operation (45) can be used. Furthermore, since the bit-streams are randomly generated with stochastic number generators (SNGs), the randomness and length of the bit-streams can significantly affect the calculation accuracy (34). Therefore, the efficient utilization of SNGs and the trade-off between the bit-stream length (i.e. the accuracy) and the resource consumption need to be carefully taken into consideration.

Compared to the conventional binary computing, the major advantage of stochastic computing is the significantly lower hardware cost for a large category of arithmetic cal-

culations. The abundant area budget offers immense design space in optimizing hardware performance via exploring the tradeoffs between the area and other metrics, such as power, latency, and parallelism degree. Therefore, SC is an interesting and promising approach to implementing large-scale DCNNs.

Multiplication. Figure 4 shows the basic multiplication components in SC domain. A unipolar multiplication can be performed by an AND gate since $P(A \cdot B = 1) = P(A = 1)P(B = 1)$ (assuming independence of two random variables), and a bipolar multiplication is performed by means of a XNOR gate since $c = 2P(C = 1) - 1 = 2(P(A = 1)P(B = 1) + P(A = 0)P(B = 0)) - 1 = (2P(A = 1) - 1)(2P(B = 1) - 1) = ab$.

Addition. We consider four popular stochastic addition methods for SC-DCNNs. *1)* OR gate (Figure 5 (a)). It is the simplest method that consumes the least hardware footprint to perform an addition, but this method introduces considerable accuracy loss because the computation "logic 1 OR logic 1" only generates a single logic 1. *2)* Multiplexer (Figure 5 (b)). It uses a multiplexer, which is the most popular way to perform additions in either the unipolar or the bipolar format (6). For example, a bipolar addition is performed as $c = 2P(C = 1) - 1 = 2(1/2P(A = 1) + 1/2P(B = 1)) - 1 = 1/2(2P(A = 1) - 1) + (2P(B = 1) - 1)) = 1/2(a+b)$. *3)* Approximate parallel counter (APC) (20) (Figure 5 (c)). It counts the number of 1s in the inputs and represents the result with a binary number. This method consumes fewer logic gates compared with the conventional accumulative parallel counter (20; 33). *4)* Two-line representation of a stochastic number (43) (Figure 5 (d)). This representation consists of a magnitude stream $M(X)$ and a sign stream $S(X)$, in which 1 represents a negative bit and 0 represents a positive bit. The value of the represented stochastic number is calculated by: $x = \frac{1}{L}\sum_{i=0}^{L-1}(1 - 2S(X_i))M(X_i)$, where L is the length of the bit-stream. As an example, -0.5 can be represented by $M(-0.5) : 10110001$ and $S(-0.5) : 11111111$. Figure 5 (d) illustrates the structure of the two-line representation-based adder. The summation of A_i (consisting of $S(A_i)$ and $M(A_i)$) and B_i are sent to a truth table, then the truth table and the counter together determine the carry bit and C_i. The truth table can be found in (43).

Hyperbolic Tangent (tanh). The tanh function is highly suitable for SC-based implementations because *i)* it can be easily implemented with a K-state finite state machine (FSM) in the SC domain (6; 28) and costs less hardware when compared to the piecewise linear approximation (PLAN)-based implementation (24) in conventional computing domain; and *ii)* replacing ReLU or sigmoid function by tanh function does not cause accuracy loss in DCNN (23). Therefore, we choose tanh as the activation function in SC-DCNNs in our design. The diagram of the FSM is shown in Figure 6. It reads the input bit-stream bit by bit, when the current input bit is one, it moves to the next state, otherwise it moves to the previous state. It outputs a

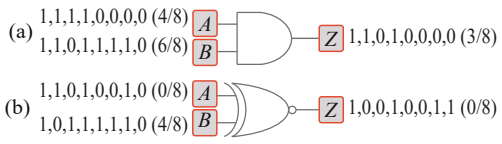

Figure 4. Stochastic multiplication. (a) Unipolar multiplication and (b) bipolar multiplication.

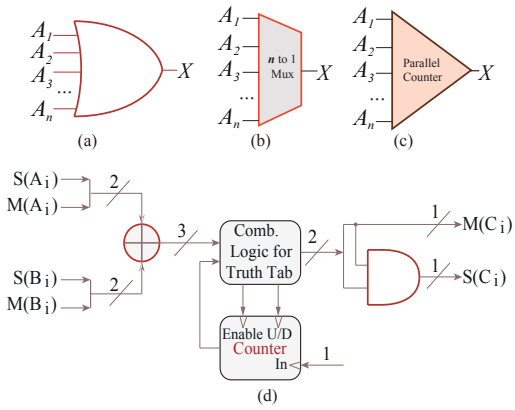

Figure 5. Stochastic addition. (a) OR gate, (b) MUX, (c) APC, and (d) two-line representation-based adder.

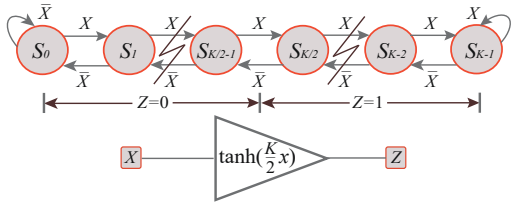

Figure 6. Stochastic hyperbolic tangent.

0 when the current state is on the left half of the diagram, otherwise it outputs a 1. The value calculated by the FSM satisfies $Stanh(K, x) \cong tanh(\frac{K}{2}x)$, where $Stanh$ denotes stochastic tanh.

3.3 Application-level vs. Hardware Accuracy

The overall network accuracy (e.g., the overall recognition or classification rates) is one of the key optimization goals of the SC-based hardware DCNN. Due to the inherent stochastic nature, the SC-based function blocks and feature extraction blocks exhibit certain degree of hardware inaccuracy. The network accuracy and hardware accuracy are different but correlated, — the high accuracy in each function block will likely lead to a high overall network accuracy. Therefore, the hardware accuracy can be optimized in the design of SC-based function blocks and feature extraction blocks.

4. Design and Optimization for Function Blocks and Feature Extraction Blocks in SC-DCNN

In this section, we first perform comprehensive designs and optimizations in order to derive the most efficient SC-based implementations for function blocks, including inner product/convolution, pooling, and activation function. The goal is to reduce power, energy and hardware resource while still maintaining high accuracy. Based on the detailed analysis of pros and cons of each basic function block design, we propose the optimized designs of feature extraction blocks for SC-DCNNs through both analysis and experiments.

4.1 Inner Product/Convolution Block Design

As shown in Figure 3 (a), an inner product/convolution block in DCNNs is composed of multiplication and addition operations. In SC-DCNNs, inputs are distributed in the range of [-1, 1], we adopt the bipolar multiplication implementation (i.e. XNOR gate) for the inner product block design. The summation of all products is performed by the adder(s). As discussed in Section 3.2, the addition operation has different implementations. To find the best option for DCNN, we replace the summation unit in Figure 3 (a) with the four different adder implementations shown in Figure 5.

OR Gate-Based Inner Product Block Design. Performing addition using OR gate is straightforward. For example, $\frac{3}{8} + \frac{4}{8}$ can be performed by "00100101 OR 11001010", which generates "11101111" ($\frac{7}{8}$). However, if first input bit-stream is changed to "10011000", the output of OR gate becomes "11011010" ($\frac{5}{8}$). Such inaccuracy is introduced by the multiple representations of the same value in SC domain and the fact that the simple "logic 1 OR logic 1" cannot tolerate such variance. To reduce the accuracy loss, the input streams should be pre-scaled to ensure that there are only very few 1's in the bit-streams. For the unipolar format bit-streams, the scaling can be easily done by dividing the original number by a scaling factor. Nevertheless, in the scenario of bipolar encoding format, there are about 50% 1's in the bit-stream when the original value is close to 0. It renders the scaling ineffective in reducing the number of 1's in the bit-stream.

Table 1 shows the average inaccuracy (absolute error) of OR gate-based inner product block with different input sizes, in which the bit-stream length is fixed at 1024 and all average inaccuracy values are obtained with the most suitable pre-scaling. The experimental results suggest that the accuracy of unipolar calculations may be acceptable, but the accuracy is too low for bipolar calculations and becomes even worse with the increased input size. Since it is almost impossible to have only positive input values and weights, we conclude that the OR gate-based inner product block is not appropriate for SC-DCNNs.

MUX-Based Inner Product Block Design. According to (6), an n-to-1 MUX can sum all inputs together and generate an output with a scaling down factor $\frac{1}{n}$. Since only one bit is selected among all inputs to that MUX at one time, the probability of each input to be selected is $\frac{1}{n}$. The selection signal is controlled by a randomly generated natural number between 1 and n. Taking Figure 3 (a) as an example, the output of the summation unit (MUX) is $\frac{1}{n}(x_0w_0 + ... + x_{n-1}w_{n-1})$.

Table 1. Absolute Errors of OR Gate-Based Inner Product Block

Input Size	16	32	64
Unipolar inputs	0.47	0.66	1.29
Bipolar inputs	1.54	1.70	2.3

Table 2. Absolute Errors of MUX-Based Inner Product Block

Input size	Bit stream length			
	512	1024	2048	4096
16	0.54	0.39	0.28	0.21
32	1.18	0.77	0.56	0.38
64	2.35	1.58	1.19	0.79

Table 2 shows the average inaccuracy (absolute error) of the MUX-based inner product block measured with different input sizes and bit-stream lengths. The accuracy loss of MUX-based block is mainly caused by the fact that only one input is selected at one time, and all the other inputs are not used. The increasing input size causes accuracy reduction because more bits are dropped. However, we see that sufficiently good accuracy can still be obtained by increasing the bit-stream length.

APC-Based Inner Product Block. The structure of a 16-bit APC is shown in Figure 7. $A_0 - A_7$ and $B_0 - B_7$ are the outputs of XNOR gates, i.e., the products of inputs x_i's and weights w_i's. Suppose the number of inputs is n and the length of a bit-stream is m, then the products of x_i's and w_i's can be represented by a bit-matrix of size $n \times m$. The function of the APC is to count the number of ones in each column and represent the result in the binary format. Therefore, the number of outputs is $\log_2 n$. Taking the 16-bit APC as an example, the output should be 4-bit to represent a number between 0 - 16. However, it is worth noting that the weight of the least significant bit is 2^1 rather than 2^0 to represent 16. Therefore, the output of the APC is a bit-matrix with size of $\log_2 n \times m$.

From Table 3, we see that the APC-based inner product block only results in less than 1% accuracy degradation when compared with the conventional accumulative parallel counter, but it can achieve about 40% reduction of gate count (20). This observation demonstrates the significant advantage of implementing efficient inner product block using APC-based method, in terms of power, energy, and hardware resource.

Two-Line Representation-Based Inner Product Block. The two-line representation-based SC scheme (43) can be used to construct a non-scaled adder. Figure 5 (d) illustrates

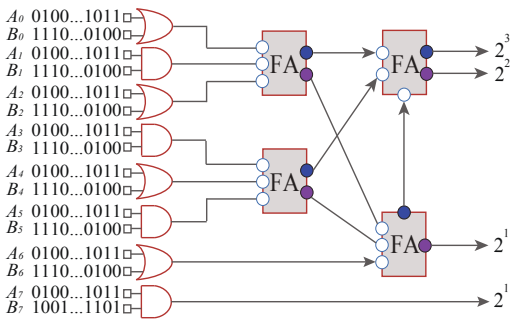

Figure 7. 16-bit Approximate Parallel Counter.

Table 3. Relative Errors of the APC-Based Compared with the Conventional Parallel Counter-Based Inner Product Blocks

Input size	Bit stream length			
	128	256	384	512
16	1.01%	0.87%	0.88%	0.84%
32	0.70%	0.61%	0.58%	0.57%
64	0.49%	0.44%	0.44%	0.42%

the structure of a two-line representation-based adder. Since A_i, B_i, and C_i are bounded as the element of $\{-1, 0, 1\}$, a carry bit may be missed. Therefore, a three-state counter is used here to store the positive or negative carry bit.

However, there are two limitations for the two-line representation-based inner product block in hardware DCNNs: *i)* An inner product block generally has more than two inputs, the overflow may often occur in the two-line representation-based inner product calculation due to its non-scaling characteristics. This leads to significant accuracy loss; and *ii)* the area overhead is too high compared with other inner product implementation methods.

4.2 Pooling Block Designs

Pooling (or down-sampling) operations are performed by pooling function blocks in DCNNs to significantly reduce *i)* inter-layer connections; and *ii)* the number of parameters and computations in the network, meanwhile maintaining the translation invariance of the extracted features (1). Average pooling and max pooling are two widely used pooling strategies. Average pooling is straightforward to implement in SC domain, while max pooling, which exhibits higher performance in general, requires more hardware resources. In order to overcome this challenge, we propose a novel hardware-oriented max pooling design with high performance and amenable to SC-based implementation.

Average Pooling Block Design. Figure 3 (b) shows how the feature map is average pooled with 2×2 filters. Since average pooling calculates the mean value of entries in a small matrix, the inherent down-scaling property of the MUX can be utilized. Therefore, the average pooling can be performed by the structure shown in Figure 5 (b) with low hardware cost.

Hardware-Oriented Max Pooling Block Design. The max pooling operation has been recently shown to provide higher performance in practice compared with the average pooling operation (1). However, in SC domain, we can find out the bit-stream with the maximum value among four candidates only after counting the total number of 1's through the whole bit-streams, which inevitably incurs long latency and considerable energy consumption.

To mitigate the cost, we propose a novel SC-based hardware-oriented max pooling scheme. The insight is that once a set of bit-streams are sliced into segments, the globally largest bit-stream (among the four candidates) has the highest probability to be the locally largest one in each set of bit-stream segments. This is because all 1's are randomly distributed in the stochastic bit-streams.

Consider the input bit-streams of the hardware-oriented max pooling block as a bit matrix. Suppose there are four bit-streams, and each has m bits, thus the size of the bit matrix is $4\times m$. Then the bit matrix is evenly sliced into small matrices whose size are $c \times m$ (i.e., each bit-stream is evenly sliced into segments whose length are c). Since the bit-streams are randomly generated, ideally, the largest row (segment) among the four rows in each small matrix is also the largest row of the global matrix. To determine the largest row in a small matrix, the number of 1s are counted in all rows in a small matrix in parallel. The maximum counted result determines the next c-bit row that is sent to the output of the pooling block. In another word, the currently selected c-bit segment is determined by the counted results of the previous matrix. To reduce latency, the c-bit segment from the first small matrix is randomly chosen. This strategy incurs zero extra latency but only causes a *negligible accuracy loss* when c is properly selected.

Figure 8 illustrates the structure of the hardware-oriented max pooling block, where the output from max_output approximately is equal to the largest bit-stream. The four input bit-streams sent to the multiplexer are also connected to four counters, and the outputs of the counters are connected to a comparator to determine the largest segment. The output of the comparator is used to control the selection of the four-to-one MUX. Suppose in the previous small bit matrix, the second row is the largest, then MUX will output the second row of the current small matrix as the current c-bit output.

Table 4 shows the result deviations of the hardware-oriented max pooling design compared with the software-based max pooling implementation. The length of a bit-stream segment is 16. In general, the proposed pooling block can provide a sufficiently accurate result even with large input size.

4.3 Activation Function Block Designs

Stanh. (6) proposed a K-state FSM-based design (i.e., Stanh) in the SC domain for implementing the tanh function and describes the relationship between Stanh and tanh as $Stanh(K, x) \cong tanh(\frac{K}{2}x)$. When the input stream x is

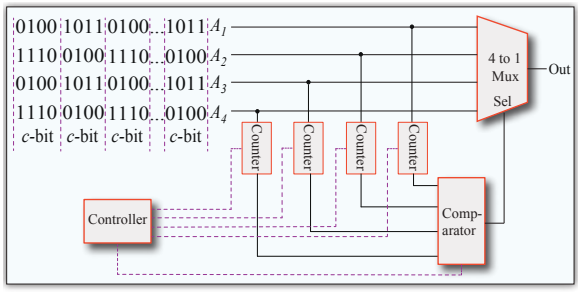

Figure 8. The Proposed Hardware-Oriented Max Pooling.

Table 4. Relative Result Deviation of Hardware-Oriented Max Pooling Block Compared with Software-Based Max Pooling

Input size	Bit-stream length			
	128	256	384	512
4	0.127	0.081	0.066	0.059
9	0.147	0.099	0.086	0.074
16	0.166	0.108	0.097	0.086

Table 5. The Relationship Between State Number and Relative Inaccuracy of Stanh

State Number	8	10	12	14	16	18	20
Relative Inaccuracy (%)	10.06	8.27	7.43	7.36	7.51	8.07	8.55

distributed in the range [-1, 1] (i.e. $\frac{K}{2}x$ is distributed in the range $[-\frac{K}{2}, \frac{K}{2}]$), this equation works well, and higher accuracy can be achieved with the increased state number K.

However, $Stanh$ cannot be applied directly in SC-DCNN for three reasons. First, as shown in Figure 9 and Table 5 (with bit-stream length fixed at 8192), when the input variable of Stanh (i.e. $\frac{K}{2}x$) is distributed in the range [-1, 1], the inaccuracy is quite notable and is not suppressed with the increasing of K. Second, the equation works well when x is precisely represented. However, when the bit-stream is not impractically long (less than 2^{16} according to our experiments), the equation should be adjusted with a consideration of bit-stream length. Third, in practice, we usually need to proactively down-scale the inputs since a bipolar stochastic number cannot reach beyond the range [-1, 1]. Moreover, the stochastic number may be sometimes passively down-scaled by certain components, such as a MUX-based adder or an average pooling block (30; 29). Therefore, a *scaling-back* process is imperative to obtain an accurate result. Based on the these reasons, the design of Stanh needs to be optimized together with other function blocks to achieve high accuracy for different bit-stream lengths and meanwhile provide a scaling-back function. More details are discussed in Section 4.4.

Btanh. Btanh is specifically designed for the APC-based adder to perform a scaled hyperbolic tangent function. Instead of using FSM, a saturated up/down counter is used to

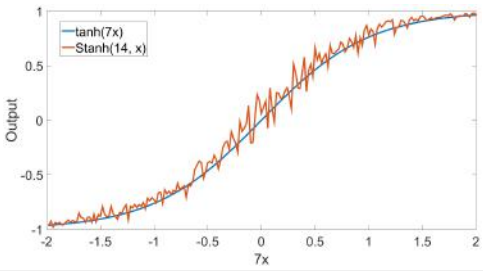

Figure 9. Output comparison of Stanh vs tanh.

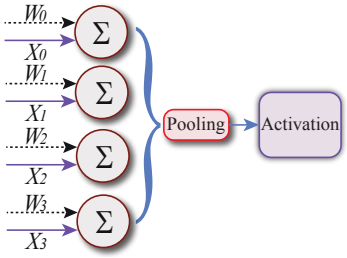

Figure 10. The structure of a feature extraction block.

convert the binary outputs of the APC-based adder back to a bit-stream. The implementation details and how to determine the number of states can be found in (21).

4.4 Design & Optimization for Feature Extraction Blocks

In this section, we propose an optimized feature extraction blocks. Based on the previous analysis and results, we select several candidates for constructing feature extraction blocks shown in Figure 10, including: the MUX-based and APC-based inner product/convolution blocks, average pooling and hardware-oriented max pooling blocks, and Stanh and Btanh blocks.

In SC domain, the parameters such as input size, bit-stream length, and the inaccuracy introduced by the previous connected block can *collectively* affect the overall performance of the feature extraction block. Therefore, the isolated optimizations on each individual basic function block are insufficient to achieve the satisfactory performance for the entire feature extraction block. For example, the most important advantage of the APC-based inner product block is its high accuracy and thus the bit-stream length can be reduced. On the other side, the most important advantage of MUX-based inner product block is the low hardware cost and the accuracy can be improved by increasing the bit-stream length. Accordingly, to achieve good performance, we cannot simply compose these basic function blocks, instead, a series of joint optimizations are performed on each type of feature extraction block. Specifically, we attempt to fully making use of the advantages of each of the building blocks.

In the following discussion, we use MUX/APC to denote the MUX-based or APC-based inner product/convolution

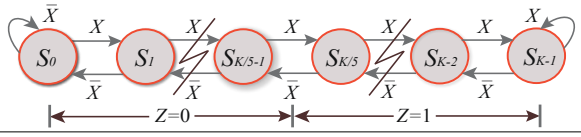

Figure 11. Structure of optimized Stanh for MUX-Max-Stanh.

blocks; use Avg/Max to denote the average or hardware-oriented max pooling blocks; use Stanh/Btanh to denote the corresponding activation function blocks. A feature extraction block configuration is represented by choosing various combinations from the three components. For example, MUX-Avg-Stanh means that four MUX-based inner product blocks, one average pooling block, and one Stanh activation function block are cascade-connected to construct an instance of feature extraction block.

MUX-Avg-Stanh. As discussed in Section 4.3, when Stanh is used, the number of states needs to be carefully selected with a comprehensive consideration of the scaling factor, bit-stream length, and accuracy requirement. Below is the empirical equation that is extracted from our comprehensive experiments to obtain the approximately optimal state number K to achieve a high accuracy:

$$K = f(L, N) \approx 2 \times \log_2 N + \frac{\log_2 L \times N}{\alpha \times \log_2 N}, \quad (1)$$

where the nearest even number to the result calculated by the above equation is assigned to K, N is the input size, L is the bit-stream length, and empirical parameter $\alpha = 33.27$.

MUX-Max-Stanh. The hardware-oriented max pooling block shown in Figure 8 in most cases generates an output that is slightly less than the maximum value. In this design of feature extraction block, the inner products are all scaled down by a factor of n (n is the input size), and the subsequent scaling back function of Stanh will enlarge the inaccuracy, especially when the positive/negative sign of the selected maximum inner product value is changed. For example, 505/1000 is a positive number, and 1% under-counting will lead the output of the hardware-oriented max pooling unit to be 495/1000, which is a negative number. Thereafter, the obtained output of Stanh may be -0.5, but the expected result should be 0.5. Therefore, the bit-stream has to be long enough to diminish the impact of under-counting, and the Stanh needs to be re-designed to fit the correct (expected) results. As shown in Figure 11, the re-designed FSM for Stanh will output 0 when the current state is at the left 1/5 of the diagram, otherwise it outputs a 1. The optimal state number K is calculated through the following empirical equation derived from experiments:

$$K = f(L, N) \approx 2 \times (\log_2 N + \log_2 L) - \frac{\alpha}{\log_2 N} - \frac{\beta}{\log_5 L}, \quad (2)$$

where the nearest even number to the result calculated by the above equation is assigned to K, N is the input size, L is the bit-stream length, $\alpha = 37$, and empirical parameter $\beta = 16.5$.

APC-Avg-Btanh. When the APC is used to construct the inner product block, conventional arithmetic calculation components, such as full adders and dividers, can be utilized to perform the averaging calculation, because the output of APC-based inner product block is a binary number. Since the design of Btanh initially aims at directly connecting to the output of APC, and an average pooling block is now inserted between APC and Btanh, the original formula proposed in (21) for calculating the optimal state number of Btanh needs to be re-formulated as:

$$K = f(N) \approx \frac{N}{2}, \quad (3)$$

from our experiments. In this equation N is the input size, and the nearest even number to $\frac{N}{2}$ is assigned to K.

APC-Max-Btanh. Although the output of APC-based inner product block is a binary number, the conventional binary comparator cannot be directly used to perform max pooling. This is because the output sequence of APC-based inner product block is still a stochastic bit-stream. If the maximum binary number is selected at each time, the pooling output is always greater than the actual maximum inner product result. Instead, the proposed hardware-oriented max pooling design should be used here, and the counters should be replaced by accumulators for accumulating the binary numbers. Thanks to the high accuracy provided by accumulators in selecting the maximum inner product result, the original Btanh design presented in (21) can be directly used without adjustment.

5. Weight Storage Scheme and Optimization

As discussed in Section 4, 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 input by users, and w_i's are weights obtained by training using software and should be stored in the hardware-based DCNNs. *Static random access memory (SRAM)* is the most suitable circuit structure 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) consumption. In this section, we present optimization techniques including efficient filter-aware SRAM sharing, weight storage method, and layer-wise weight storage optimizations.

5.1 Efficient Filter-Aware SRAM Sharing Scheme

Since all receptive fields of a feature map share one filter (a matrix of weights), all weights functionally can be separated into filter-based blocks, and each weight block is shared by all inner product/convolution blocks using the corresponding filter. Inspired by this fact, we propose an efficient filter-aware SRAM sharing scheme, with structure illustrated in Figure 12. The scheme divides the whole SRAM into small blocks to mimic filters. Besides, all inner product blocks can also be separated into feature map-based groups, where each group extracts a specific feature map. In this scheme, a local SRAM block is shared by all the inner product blocks of the corresponding group. The weights of the corresponding

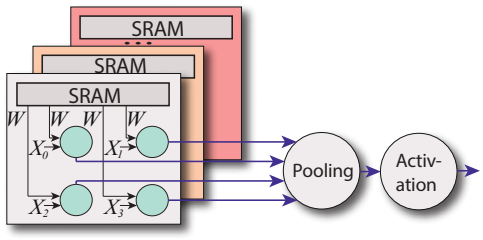

Figure 12. Filter-Aware SRAM Sharing Scheme.

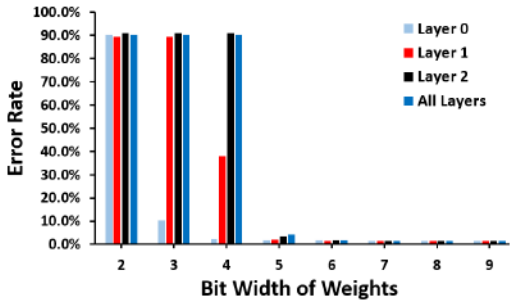

Figure 13. The impact of precision of weights at different layers on the overall SC-DCNN network accuracy.

filter are stored in the local SRAM block of this group. This scheme significantly reduces the routing overhead and wire delay.

5.2 Weight Storage Method

Besides the reduction on routing overhead, the size of SRAM blocks can also be reduced by trading off accuracy for less hardware resources. The trade-off is realized by eliminating certain least significant bits of a weight value to reduce the SRAM size. In the following, we present a weight storage method that significantly reduces the SRAM size with little network accuracy loss.

Baseline: High Precision Weight Storage. In general, DCNNs are trained with single floating point precision. In hardware, the SRAM up to 64-bit is needed for storing one weight value in the fixed-point format to maintain its original high precision. This scheme provides high accuracy since there is almost no information loss of weights. However, it also incurs high hardware consumption in that the size of SRAM and its related read/write circuits increase with the precision increase of the stored weight values.

Low Precision Weight Storage Method. According to our application-level experiments, many least significant bits that are far from the decimal point only have a very limited impact on the overall accuracy. Therefore, the number of bits for weight representation in the SRAM block can be significantly reduced. We propose 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), then the binary number to be stored for representing x is: $y = \frac{Int(\frac{x+1}{2} \times 2^w)}{2^w}$, where $Int()$ means only keeping the integer part. Figure 13 illustrates the network error rates when the reductions of weights' precision are performed at a single layer or all layers. The precision loss of weights at Layer0 (consisting of a convolutional layer and pooling layer) has the least impact, while the precision loss of weights at Layer2 (a fully connected layer) has the most significant impact. The reason is that Layer2 is the fully connected layer that has the largest number of weights. On the other hand, when w is set equal to or greater than seven, the network error rates are low enough and almost do not decrease with the further precision increase. Therefore, the proposed weight storage method can significantly reduce the size of SRAMs and their read/write circuits by reducing precision. The area savings estimated using CACTI 5.3 (42) is 10.3×.

5.3 Layer-wise Weight Storage Optimization

As shown in Figure 13, the precision of weights at different layers have different impacts on the overall accuracy of the network. (18) presented a method that set different weight precisions at different layers to save weight storage. In SC-DCNN, we adopt the same strategy to improve the hardware performance. Specifically, this method is effective to obtain savings in SRAM area and power (energy) consumption because Layer2 has the most number of weights compared with the previous layers. For instance, when we set weights as 7-7-6 at the three layers of LeNet5, the network error rate is 1.65%, which has only 0.12% accuracy degradation compared with the error rate obtained using software-only implementation. However, 12× improvements on area and 11.9× improvements on power consumption are achieved for the weight representations (from CACTI 5.3 estimations), compared with the baseline without any reduction in weight representation bits.

6. Overall SC-DCNN Optimizations and Results

In this section, we present optimizations of feature extraction blocks along with comparison results with respect to accuracy, area/hardware footprint, power (energy) consumption, etc. Based on the results, we perform thorough optimizations on the overall SC-DCNN to construct LeNet5 structure, which is one of the most well-known large-scale deep DCNN structure. The goal is to minimize area and power (energy) consumption while maintaining a high network accuracy. We present comprehensive comparison results among *i)* SC-DCNN designs with different target network accuracy, and *ii)* existing hardware platforms. The hardware performance of the various SC-DCNN implementations regarding area, path delay, power and energy consumption are obtained by: *i)* synthesizing with the 45nm

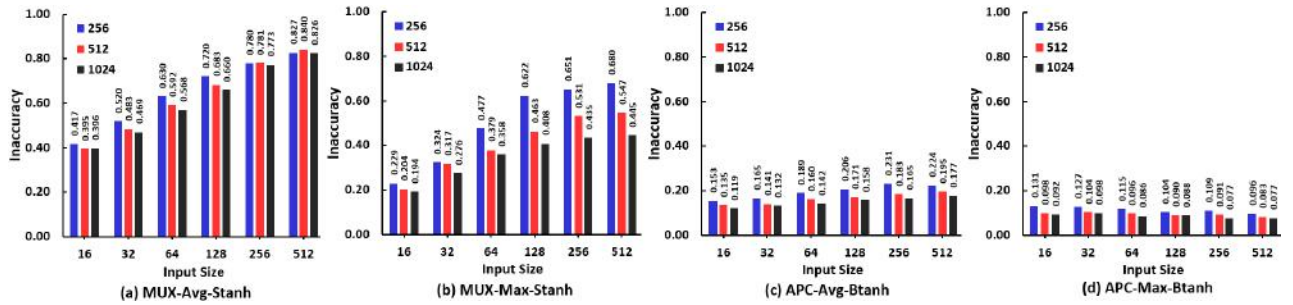

Figure 14. Input size versus absolute inaccuracy for (a) MUX-Avg-Stanh, (b) MUX-Max-Stanh, (c) APC-Avg-Btanh, and (d) APC-Max-Btanh with different bit stream lengths.

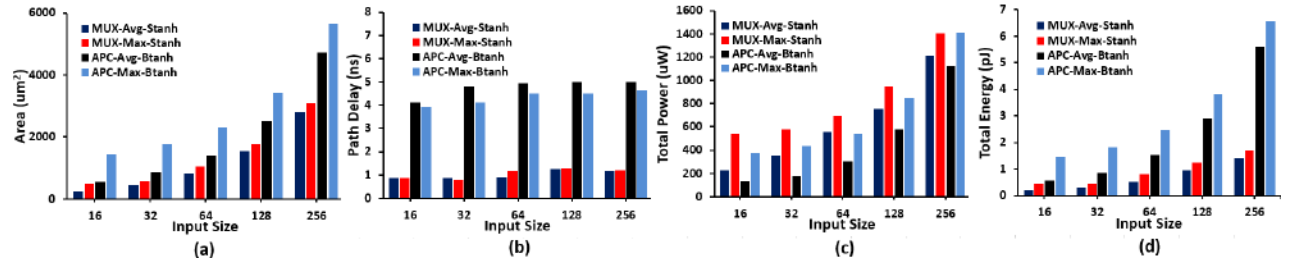

Figure 15. Input size versus (a) area, (b) path delay, (c) total power, and (d) total energy for four different designs of feature extraction blocks.

Nangate Open Cell Library (3) using Synopsys Design Compiler; and *ii)* estimating using CACTI 5.3 (42) for the SRAM blocks. The key peripheral circuitries in the SC domain (e.g. the random number generators) are developed using the design in (22) and synthesized using Synopsys Design Compiler.

6.1 Optimization Results on Feature Extraction Blocks

We present optimization results of feature extraction blocks with different structures, input sizes, and bit-stream lengths on accuracy, area/hardware footprint, power (energy) consumption, etc. Figure 14 illustrates the accuracy results of four types of feature extraction blocks: MUX-Avg-Stanh, MUX-Max-Stanh, APC-Avg-Btanh, and APC-Max-Btanh. The horizontal axis represents the input size that increases logarithmically from 16 (2^4) to 256 (2^8). The vertical axis represents the hardware inaccuracy of feature extraction blocks. Three bit-stream lengths are tested and their impacts are shown in the figure. Figure 15 illustrates the comparisons among four feature extraction blocks with respect to area, path delay, power, and energy consumption. The horizontal axis represents the input size that increases logarithmically from 16 (2^4) to 256 (2^8). The bit-stream length is fixed at 1024.

MUX-Avg-Stanh. From Figure 14 (a), we see that it has the worst accuracy performance among the four structures. It is because the MUX-based adder, as mentioned in Section 4, is a down-scaling adder and incurs inaccuracy due to information loss. Moreover, average pooling is performed with MUXes, thus the inner products are further down-scaled and more inaccuracy is incurred. As a result, this structure of feature extraction block is only appropriate for dealing with receptive fields with a small size. On the other hand, it is the most area and energy efficient design with the smallest path delay. Hence, it is appropriate for scenarios with tight limitations on area and delay.

MUX-Max-Stanh. Figure 14 (b) shows that it has a better accuracy compared with MUX-Avg-Stanh. The reason is that the mean of four numbers is generally closer to zero than the maximum value of the four numbers. As discussed in Section 4, minor inaccuracy on the stochastic numbers near zero can cause significant inaccuracy on the outputs of feature extraction blocks. Thus the structures with hardware-oriented pooling are more resilient than the structures with average pooling. In addition, the accuracy can be significantly improved by increasing the bit-stream length, thus this structure can be applied for dealing with the receptive fields with both small and large sizes. With respect to area, path delay, and energy, its performance is a close second to the MUX-Avg-Stanh. Despite its relatively high power consumption, the power can be remarkably reduced by trading off the path delay.

APC-Avg-Btanh. Figure 14 (c) and 14 (d) illustrate the hardware inaccuracy of APC-based feature extraction blocks. The results imply that they provide significantly higher accuracy than the MUX-based feature extraction blocks. It is because the APC-based inner product blocks maintain most information of inner products and thus generate results with high accuracy. It is exactly the drawback of the MUX-based inner product blocks. On the other

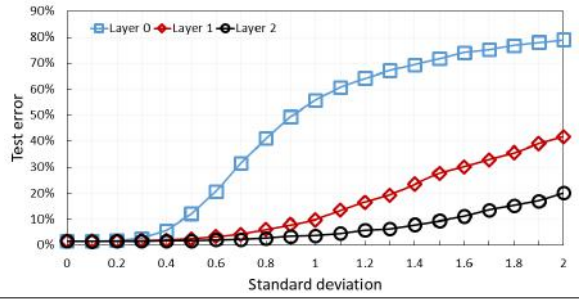

Figure 16. The impact of inaccuracies at each layer on the overall SC-DCNN network accuracy.

hand, APC-based feature extraction blocks consume more hardware resources and result in much longer path delays and higher energy consumption. The long path delay is partially the reason why the power consumption is lower than MUX-based designs. Therefore, the APC-Avg-Btanh is appropriate for DCNN implementations that have a tight specification on the accuracy performance and have a relative loose hardware resource constraint.

APC-Max-Btanh. Figure 14 (d) indicates that this feature extraction block design has the best accuracy due to several reasons. First, it is an APC-based design. Second, the average pooling in the APC-Avg-Btanh causes more information loss than the proposed hardware-oriented max pooling. To be more specific, the fractional part of the number after average pooling is dropped: the mean of (2, 3, 4, 5) is 3.5, but it will be represented as 3 in binary format, thus some information is lost during the average pooling. Generally, the increase of input size will incur significant inaccuracy except for APC-Max-Btanh. The reason that APC-Max-Btanh performs better with more inputs is: more inputs will make the four inner products sent to the pooling function block more distinct from one another, in another word, more inputs result in higher accuracy in selecting the maximum value. The drawbacks of APC-Max-Btanh are also different. It has the highest area and energy consumption, and its path delay is just second (but very close) to APC-Avg-Btanh. Also, its power consumption is second (but close) to the MUX-Max-Stanh. Accordingly, this design is appropriate for the applications that have a very tight requirement on the accuracy.

6.2 Layer-wise Feature Extraction Block Configurations

Inspired by the experiment results of the layer-wise weight storage scheme, we also investigate the sensitivities of different layers to the inaccuracy. Figure 16 illustrates that different layers have different error sensitivities. Combining this observation with the observations drawn from Figure 14 and Figure 15, we propose a layer-wise configuration strategy that uses different types of feature extraction blocks in different layers to minimize area and power (energy) consumption while maintaining a high network accuracy.

6.3 Overall Optimizations and Results on SC-DCNNs

Using the design strategies presented so far, we perform holistic optimizations on the overall SC-DCNN to construct the LeNet 5 DCNN structure. The (max pooling-based or average pooling-based) LeNet 5 is a widely-used DCNN structure (26) with a configuration of 784-11520-2880-3200-800-500-10. The SC-DCNNs are evaluated with the MNIST handwritten digit image dataset (9), which consists of 60,000 training data and 10,000 testing data.

The baseline error rates of the max pooling-based and average pooling-based LeNet5 DCNNs using software implementations are 1.53% and 2.24%, respectively. In the optimization procedure, we set 1.5% as the threshold on the error rate difference compared with the error rates of software implementation. In another word, the network accuracy degradation of the SC-DCNNs cannot exceed 1.5%. We set the maximum bit-stream length as 1024 to avoid excessively long delays. In the optimization procedure, for the configurations that achieve the target network accuracy, the bit-stream length is reduced by half in order to reduce energy consumption. Configurations are removed if they fail to meet the network accuracy goal. The process is iterated until no configuration is left.

Table 6 displays some selected typical configurations and their comparison results (including the consumption of SRAMs and random number generators). Configurations No.1-6 are max pooling-based SC-DCNNs, and No.7-12 are average pooling-based SC-DCNNs. It can be observed that the configurations involving more MUX-based feature extraction blocks achieve lower hardware cost. Those involving more APC-based feature extraction blocks achieve higher accuracy. For the max pooling-based configurations, No.1 is the most area efficient as well as power efficient configuration, and No.5 is the most energy efficient configuration. With regard to the average pooling-based configurations, No.7, 9, 11 are the most area efficient and power efficient configurations, and No.11 is the most energy efficient configuration.

Table 7 shows the results of two configurations of our proposed SC-DCNNs together with other software and hardware platforms. It includes software implementations using Intel Xeon Dual-Core W5580 and Nvidia Tesla C2075 GPU and five hardware platforms: Minitaur (32), SpiNNaker (39), TrueNorth (11; 12), DaDianNao (7), and EIE-64PE (14). EIE's performance was evaluated on a fully connected layer of AlexNet (23). The state-of-the-art platform DaDianNao proposed an ASIC "node" that could be connected in parallel to implement a large-scale DCNN. Other hardware platforms implement different types of hardware neural networks such as spiking neural network or deep-belief network.

For SC-DCNN, the configuration No.6 and No.11 are selected to compare with software implementation on

[1]ANN: Artificial Neural Network; [2]DBN: Deep Belief Network; [3]SNN: Spiking Neural Network

Table 6. Comparison among Various SC-DCNN Designs Implementing LeNet 5

No.	Pooling	Bit Stream	Configuration			Performance				
			Layer 0	Layer 1	Layer 2	Inaccuracy (%)	Area (mm^2)	Power (W)	Delay (ns)	Energy (μJ)
1	Max	1024	MUX	MUX	APC	2.64	19.1	1.74	5120	8.9
2			MUX	APC	APC	2.23	22.9	2.13	5120	10.9
3		512	APC	MUX	APC	1.91	32.7	3.14	2560	8.0
4			APC	APC	APC	1.68	36.4	3.53	2560	9.0
5		256	APC	MUX	APC	2.13	32.7	3.14	1280	4.0
6			APC	APC	APC	1.74	36.4	3.53	1280	4.5
7	Average	1024	MUX	APC	APC	3.06	17.0	1.53	5120	7.8
8			APC	APC	APC	2.58	22.1	2.14	5120	11.0
9		512	MUX	APC	APC	3.16	17.0	1.53	2560	3.9
10			APC	APC	APC	2.65	22.1	2.14	2560	5.5
11		256	MUX	APC	APC	3.36	17.0	1.53	1280	2.0
12			APC	APC	APC	2.76	22.1	2.14	1280	2.7

Table 7. List of Existing Hardware Platforms

Platform	Dataset	Network Type	Year	Platform Type	Area (mm^2)	Power (W)	Accuracy (%)	Throughput (Images/s)	Area Efficiency (Images/s/mm^2)	Energy Efficiency (Images/J)
SC-DCNN (No.6)	MNIST	CNN	2016	ASIC	36.4	3.53	98.26	781250	21439	221287
SC-DCNN (No.11)			2016	ASIC	17.0	1.53	96.64	781250	45946	510734
2×Intel Xeon W5580			2009	CPU	263	156	98.46	656	2.5	4.2
Nvidia Tesla C2075			2011	GPU	520	202.5	98.46	2333	4.5	3.2
Minitaur (32)		ANN[1]	2014	FPGA	N/A	≤1.5	92.00	4880	N/A	≥3253
SpiNNaker (39)		DBN[2]	2015	ARM	N/A	0.3	95.00	50	N/A	166.7
TrueNorth (11; 12)		SNN[3]	2015	ASIC	430	0.18	99.42	1000	2.3	9259
DaDianNao (7)	ImageNet	CNN	2014	ASIC	67.7	15.97	N/A	147938	2185	9263
EIE-64PE (14)		CNN layer	2016	ASIC	40.8	0.59	N/A	81967	2009	138927

CPU server or GPU. No.6 is selected because it is the most accurate max pooling-based configuration. No.11 is selected because it is the most energy efficient average pooling-based configuration. According to Table 7, the proposed SC-DCNNs are much more area efficient: the area of Nvidia Tesla C2075 is up to 30.6× of the area of SC-DCNN (No.11). Moreover, our proposed SC-DCNNs also have outstanding performance in terms of throughput, area efficiency, and energy efficiency. Compared with Nvidia Tesla C2075, the SC-DCNN (No.11) achieves 15625× throughput improvements and 159604× energy efficiency improvements.

Although LeNet5 is relatively small, it is still a representative DCNN. Most of the computation blocks of LeNet5 can be applied to other networks (e.g. AlexNet). Based on our experiments, inside a network, the inaccuracy introduced by the SC-based components can significantly compensate each other. Therefore, we expect that SC-induced inaccuracy will be further reduced with larger networks. We also anticipate higher area/energy efficiency in larger DCNNs. Many of the basic computations are the same for different types of networks, thus the significant area/energy efficiency improvement in each component will result in improvement of the whole network (compared with binary designs) for different network sizes/types. In addition, when the network is larger, there are potentially more optimization space for further improving the area/energy efficiency. Therefore, we claim that the proposed SC-DCNNs have good scalability. The investigations on larger networks will be conducted in our future work.

7. Conclusion

In this paper, we propose *SC-DCNN*, the first comprehensive design and optimization framework of SC-based DCNNs. SC-DCNN fully utilizes the advantages of SC and achieves remarkably low hardware footprint, low power and energy consumption, while maintaining high network accuracy. We fully explore the design space of different components to achieve high power (energy) efficiency and low hardware footprint. First, we investigated various function blocks including inner product calculations, pooling operations, and activation functions. Then we propose four designs of feature extraction blocks, which are in charge of extracting features from input feature maps, by connecting different basic function blocks with joint optimization. Moreover, three weight storage optimization schemes are investigated for reducing the area and power (energy) consumption of SRAM. Experimental results demonstrate that our proposed SC-DCNN achieves low hardware footprint and low energy consumption. It achieves the throughput of 781250 images/s, area efficiency of 45946 images/s/mm^2, and energy efficiency of 510734 images/J.

8. Acknowledgement

We thank anonymous reviewers for their valuable feedback. 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] Stanford cs class, cs231n: Convolutional neural networks for visual recognition, 2016. URL http://cs231n.github.io/convolutional-networks/.

[2] Convolutional neural networks (lenet), 2016. URL http://deeplearning.net/tutorial/lenet.html#motivation.

[3] Nangate 45nm Open Library, Nangate Inc., 2009. URL http://www.nangate.com/.

[4] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

[5] R. Andri, L. Cavigelli, D. Rossi, and L. Benini. Yodann: An ultra-low power convolutional neural network accelerator based on binary weights. *arXiv preprint arXiv:1606.05487*, 2016.

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

[7] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. 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.

[8] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

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

[10] L. Deng and D. Yu. Deep learning. *Signal Processing*, 7:3–4, 2014.

[11] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

[12] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha. Convolutional networks for fast, energy-efficient neuromorphic computing. *CoRR*, abs/1603.08270, 2016. URL http://arxiv.org/abs/1603.08270.

[13] B. R. Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 149–156. ACM, 1967.

[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. *arXiv preprint arXiv:1602.01528*, 2016.

[15] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman. Memristor crossbar-based neuromorphic computing system: A case study. *IEEE transactions on neural networks and learning systems*, 25(10):1864–1878, 2014.

[16] 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*, pages 880–883. EDA Consortium, 2015.

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[18] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, R. Urtasun, and A. Moshovos. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint arXiv:1511.05236*, 2015.

[19] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. 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.

[20] K. Kim, J. Lee, and K. Choi. Approximate de-randomizer for stochastic circuits. *Proc. ISOCC*, 2015.

[21] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. 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.

[22] K. Kim, J. Lee, and K. 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.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] D. Larkin, A. Kinane, V. Muresan, and N. OConnor. An efficient hardware architecture for a neural network activation function generator. In *International Symposium on Neural Networks*, pages 1319–1327. Springer, 2006.

[25] E. László, P. Szolgay, and Z. 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.

[26] Y. LeCun. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 2015.

[27] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[28] J. Li, A. Ren, Z. Li, C. Ding, B. Yuan, Q. Qiu, and Y. Wang. Towards acceleration of deep convolutional neural networks using stochastic computing. In *The 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE*, 2017.

[29] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan. Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks. In *Com-*

puter Design (ICCD), 2016 IEEE 34th International Conference on, pages 678–681. IEEE, 2016.

[30] Z. Li, A. Ren, J. Li, Q. Qiu, B. Yuan, J. Draper, and Y. Wang. Structural design optimization for deep convolutional neural networks using stochastic computing. 2017.

[31] 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)*, pages 575–580. IEEE, 2016.

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

[33] B. Parhami and C.-H. Yeh. Accumulative parallel counters. In *Signals, Systems and Computers, 1995. 1995 Conference Record of the Twenty-Ninth Asilomar Conference on*, volume 2, pages 966–970. IEEE, 1995.

[34] A. Ren, Z. Li, Y. Wang, Q. Qiu, and B. Yuan. Designing reconfigurable large-scale deep learning systems using stochastic computing. In *2016 IEEE International Conference on Rebooting Computing*. IEEE, 2016.

[35] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran. Deep convolutional neural networks for lvcsr. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8614–8618. IEEE, 2013.

[36] S. Sato, K. Nemoto, S. Akimoto, M. Kinjo, and K. Nakajima. Implementation of a new neurochip using stochastic logic. *IEEE Transactions on Neural Networks*, 14(5):1122–1127, 2003.

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

[38] G. V. STOICA, R. DOGARU, and C. Stoica. High performance cuda based cnn image processor, 2015.

[39] E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.-C. Liu, and S. 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.

[40] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[41] M. Tanomoto, S. Takamaeda-Yamazaki, J. Yao, and Y. Nakashima. A cgra-based approach for accelerating convolutional neural networks. In *Embedded Multicore/Many-core Systems-on-Chip (MCSoC), 2015 IEEE 9th International Symposium on*, pages 73–80. IEEE, 2015.

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

[43] S. Toral, J. Quero, and L. Franquelo. Stochastic pulse coded arithmetic. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 1, pages 599–602. IEEE, 2000.

[44] L. Xia, B. Li, T. Tang, P. Gu, X. Yin, W. Huangfu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang. Mnsim: Simulation platform for memristor-based neuromorphic computing system. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 469–474. IEEE, 2016.

[45] B. Yuan, C. Zhang, and Z. Wang. Design space exploration for hardware-efficient stochastic computing: A case study on discrete cosine transformation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6555–6559. IEEE, 2016.

[46] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. 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.