INTEGRATION the VLSI journal xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

# Normalization and dropout for stochastic computing-based deep convolutional neural networks

Ji Li[a,*], Zihao Yuan[a], Zhe Li[b], Ao Ren[b], Caiwen Ding[b], Jeffrey Draper[a,c], Shahin Nazarian[a], Qinru Qiu[b], Bo Yuan[d], Yanzhi Wang[b]

[a] University of Southern California, Department of Electrical Engineering, Los Angeles, CA 90089, USA
[b] Syracuse University, Department of Electrical Engineering and Computer Science, Syracuse, NY 13210, USA
[c] University of Southern California, Information Sciences Institute, Marina del Rey, CA 90292, USA
[d] City University of New York, City College, Department of Electrical Engineering, New York, NY 10031, USA

ARTICLE INFO

*Keywords:*
Deep learning
Deep convolutional neural networks
Dropout
Normalization

ABSTRACT

Recently, Deep Convolutional Neural Network (DCNN) has been recognized as the most effective model for pattern recognition and classification tasks. With the fast growing Internet of Things (IoTs) and wearable devices, it becomes attractive to implement DCNNs in embedded and portable systems. However, novel computing paradigms are urgently required to deploy DCNNs that have huge power consumptions and complex topologies in systems with limited area and power supply. Recent works have demonstrated that Stochastic Computing (SC) can radically simplify the hardware implementation of arithmetic units and has the potential to bring the success of DCNNs to embedded systems. This paper introduces normalization and dropout, which are essential techniques for the state-of-the-art DCNNs, to the existing SC-based DCNN frameworks. In this work, the feature extraction block of DCNNs is implemented using an approximate parallel counter, a near-max pooling block and an SC-based rectified linear activation unit. A novel SC-based normalization design is proposed, which includes a square and summation unit, an activation unit and a division unit. The dropout technique is integrated into the training phase and the learned weights are adjusted during the hardware implementation. Experimental results on AlexNet with the ImageNet dataset show that the SC-based DCNN with the proposed normalization and dropout techniques achieves 3.26% top-1 accuracy improvement and 3.05% top-5 accuracy improvement compared with the SC-based DCNN without these two essential techniques, confirming the effectiveness of our normalization and dropout designs.

## 1. Introduction

Deep Convolutional Neural Network (DCNN) has recently achieved unprecedented success in various applications, such as image recognition [1], natural language processing [2], video recognition [3], and speech processing [4]. As DCNN breaks several long-time records in different popular datasets, it is recognized as the dominant approach for almost all pattern detection and classification tasks [5]. With the fast advancement and widespread deployment of Internet of Things (IoTs) and wearable devices [6], implementing DCNNs in embedded and portable systems is becoming increasingly attractive.

As large-scale DCNNs may use millions of neurons, the intensive computation of DCNNs inhibits their deployment from cloud clusters to local platforms. To resolve this issue, numerous hardware-based DCNNs that use General-Purpose Graphics Processing Units (GPGPUs) [7,8], FPGA [9] and ASIC [10,11] are proposed to accelerate the deep learning systems with huge power, energy and area reduction compared with software. Nevertheless, novel computing paradigms are required to make DCNNs compact enough for light-weight IoT and wearable devices with stringent power requirements.

The recent advancements [12–17] demonstrate that Stochastic Computing (SC), as a low-cost and soft error resilient alternative to conventional binary computing [18–21], can radically simplify the hardware footprint of arithmetic units in DCNNs and has the potential to satisfy the stringent power requirements in embedded devices. Stochastic designs for fundamental operations (i.e., inner product, pooling, activation, and softmax regression) of DCNNs have been proposed in [12–14,16,17], and medium scale DCNNs have been implemented in the SC regime [15].

Despite power and area efficiency achieved by the existing SC approach compared with the conventional binary approach, no prior work has investigated the two software techniques essential for large scale DCNNs:

[*] Corresponding author.
*E-mail addresses:* jli724@usc.edu (J. Li), zihaoyua@usc.edu (Z. Yuan), zheli89@syr.edu (Z. Li), aren@syr.edu (A. Ren), cading@syr.edu (C. Ding), draper@isi.edu (J. Draper), shahin@usc.edu (S. Nazarian), qiqiu@syr.edu (Q. Qiu), byuan@ccny.cuny.edu (B. Yuan), ywang393@syr.edu (Y. Wang).

https://doi.org/10.1016/j.vlsi.2017.11.002

(i) Local Response Normalization (LRN) and (ii) dropout. LRN is used to form a local maxima and increase the sensory perception [1], which is vital to state-of-the-art DCNNs, e.g., 1.2–1.4% accuracy improvement was reported in AlexNet [1]. Dropout, mitigates the overfitting problem, which results in poor DCNN performance on held-out test data when it is trained with a small set of training data [22,23]. Without a careful design of LRN and integration of dropout in the SC domain, the current SC framework developed in [20,12,15] cannot be extended to more advanced and larger-scaled DCNNs like AlexNet [1] without performance degradation.

In this paper, we integrate the LRN layer and dropout techniques into the existing SC-based DCNN frameworks. Unlike the previous studies [20,12,15], we consider max pooling and Rectified Linear Units (ReLU) in DCNNs. This is because max pooling and ReLU are more commonly applied in the state-of-the-art DCNNs with superior performance than average pooling and hyperbolic tangent activation, respectively. The basic building block in software DCNNs is Feature Extraction Block (FEB), which extracts high-level features from the raw inputs or previous low-level abstractions. Accordingly, we present an SC-based FEB design with Approximate Parallel Counter (APC)-based inner product unit, hardware-oriented max pooling, and SC-based ReLU activation. Due to the inherent stochastic nature, SC-based FEB exhibits a certain amount of imprecision. Hence, the entire FEB block is carefully optimized to achieve sufficient accuracy. In addition, we propose an optimized SC-based LRN design that is composed of division, activation, square and summation units. Finally, the dropout is inserted in the software training phase and the learned weights are adjusted during the hardware implementation.

The contributions of this work are threefold. First, we present the SC-based FEB design with max pooling and ReLU, which are widely used in most of the state-of-the-art DCNNs. The near-max pooling proposed in [15] is improved to achieve better performance. Second, we are the first to propose the stochastic LRN design for SC-based DCNNs. Third, this is the first work to integrate the dropout technique into the existing SC-based DCNN frameworks, in order to reduce the training time and mitigate the overfitting issue. Experimental results on AlexNet with the ImageNet dataset show that the SC-based DCNN with the proposed LRN and dropout techniques achieves 3.26% top-1 accuracy improvement and 3.05% top-5 accuracy improvement compared with the SC-based DCNN without these two essential techniques, demonstrating the effectiveness of proposed normalization and dropout designs.

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 gives an overview of DCNN architecture and related stochastic computing building blocks. SC-based FEB design is presented in Section 4, and Section 5 presents the proposed normalization and dropout design. Section 6 shows the experimental results and finally this paper is concluded in Section 7.

## 2. Prior work

A significant amount of research efforts have been invested in the context of accelerating neural networks using hardware. The hardware accelerators can be divided into three groups.

### 2.1. FPGA approach

Many FPGA-based accelerators [24–26] are proposed to implement DCNNs by maximizing the underlying FPGA computing and bandwidth resource utilization for operations in DCNNs. M. Motamedi et al. proposed a design space exploration algorithm for obtaining the implementation of a DCNN with the highest performance on a given FPGA platform [24]. In order overcome the challenge of designing a common architecture that can perform well for all convolutional layers, A. Rahman et al. presented a flexible and highly efficient 3D neuron array architecture for convolutional layers [25]. A hardware/software co-designed library called Caffeine was proposed by Zhang et al. to efficiently accelerate the DCNN on FPGAs with integration into the industry-standard software deep learning framework Caffe [26].

### 2.2. Conventional ASIC approach

In parallel, various ASIC-based accelerators have come into existence [10,11,27]. Chen et al. proposed the custom multi-chip machine-learning architecture DaDianNao for state-of-the-art DCNNs [10]. To improve the energy efficiency, Chen et al. presented Eyeriss with reduced data movement, skipped zeros and data compression [11]. A deep compression framework EIE was proposed by Han et al. to accelerate the sparse matrix-vector multiplication with weight sharing [27].

### 2.3. Stochastic computing approach

Stochastic computing technology has become very attractive for DCNN implementations due to its significantly reduced hardware footprint. Several fundamental SC components were designed by Brown and Card for artificial neuron networks [28]. K. Kim et al. presented an approximate parallel counter (APC) based SC neuron design for deep belief network [20]. Li et al. proposed multiplexer/APC based SC neurons for DCNNs [12]. Ren et al. developed SC-DCNN in [15], which achieved low hardware footprint and low power (energy) consumption. However, the normalization and dropout techniques, which are widely deployed in the software-based DCNNs and essential for large-scale DCNNs, have not been integrated in the SC-based hardware DCNNs.

## 3. DCNN architecture overview

### 3.1. DCNN architecture

As shown in Fig. 1, a general DCNN is composed of a stack of convolutional layers, pooling layers, and fully-connected layers. A convolutional layer is followed by a pooling layer, which extracts features from raw inputs or the previous feature maps. A fully connected layer aggregates the high level features, and a softmax regression is applied to derive the final output. The basic component of DCNN is the Feature Extraction Block (FEB), which conducts inner product, pooling and activation operations.

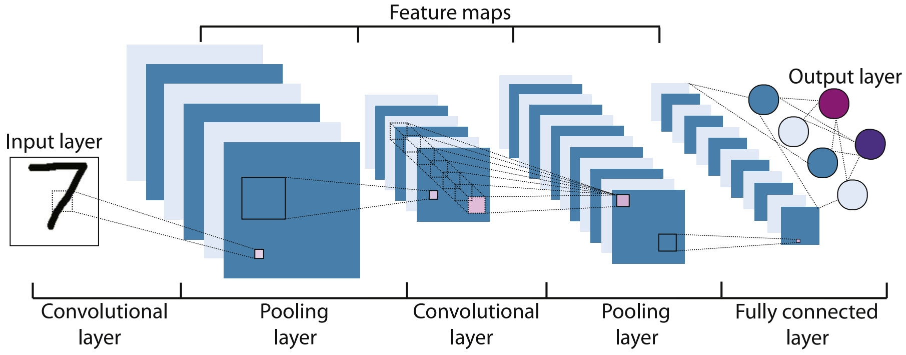

**Fig. 1.** The fifth generation of LeNet DCNN architecture.

### 3.2. Stochastic computing

In this paper, we adopt the bipolar encoding format, in order to represent negative numbers in DCNNs. In the bipolar encoding format of SC, number $x$ is represented by a bit stream $X$, i.e., $x = 2P(X = 1) - 1$. For instance, $x = 0.4$ can be represented by $X = 1101011101$. The basic arithmetic operations in DCNN are multiplication, addition, and activation. With SC, these operations can be implemented with extremely small circuits as follows.

#### 3.2.1. Stochastic multiplication

In the SC domain, bipolar multiplication is performed by an XNOR gate, as shown in Fig. 2. We denote the probabilities of 1 on the input bit streams by $P(A)$ and $P(B)$, and the output of the XNOR gate is $z = 2P(Z = 1) - 1 = 2(P(A = 1)P(B = 1) + P(A = 0)P(B = 0)) - 1 = (2P(A = 1) - 1)(2P(B = 1) - 1) = ab$ . Note that the input bit streams are assumed to be suitably uncorrelated or independent in the above calculations.

#### 3.2.2. Stochastic addition

In the bipolar encoding format, the stochastic addition can be performed by a multiplexer (MUX) [12] or an Approximate Parallel Counter (APC) [20], as shown in Figs. 3 and 4, respectively. More specifically, a MUX performs scaled addition by randomly selecting one input $i$ among $n$ inputs with probability $p_i$ such that $\sum_{i=1}^{n} p_i = 1$. An example of adding $a$ and $b$ using MUX with $p_1 = p_2 = 50\%$ is shown in Fig. 3, which performs scaled addition in bipolar format, i.e., $z = 2\cdot P(Z) - 1 = 2\cdot(\frac{P(A)}{2} + \frac{P(B)}{2}) - 1 = \frac{1}{2}\cdot((2\cdot P(A) - 1)+(2\cdot P(B) - 1)) = \frac{1}{2}\cdot(a + b)$. 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. Hence, the APC shown in Fig. 4 is proposed in [20] to improve the accuracy by gathering all the input bit streams. Considering the large FEBs in large-scale DCNNs, we adopt APC as the adder design so as to achieve sufficient accuracy. Details will be discussed in Section 4.1.

#### 3.2.3. Stochastic activation

As shown in Fig. 5, the hyperbolic tangent function (i.e., $tanh(\cdot)$) is implemented using a $K$-state FSM, in which half of the states generate output 0 and the other half states generate 1. According to [28], for a given bipolar stochastic number $x$, the result of the FSM design is a stochastic approximation to the $tanh$ function, i.e., $Stanh(K, x) = tanh(\frac{K\cdot x}{2})$. As the output of an APC adder is in binary format, this FSM design is re-designed by the authors in [20] as an up/down counter to calculate the $Btanh(\cdot)$ for binary inputs. The accuracy of the activation function is determined by state number $K$ and input stream length.

## 4. Proposed stochastic computing-based feature extraction block

The FEB considered in this work is composed of inner product, max pooling, and ReLU activation units, which is implemented by APC-based inner product, hardware-oriented max pooling, and SC-based ReLU activation, respectively.

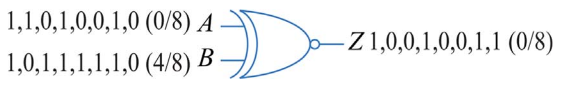

**Fig. 2.** Stochastic multiplication using bipolar encoding format.

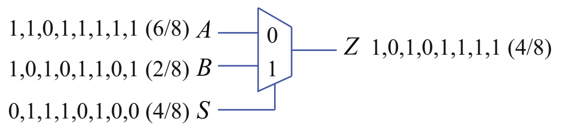

**Fig. 3.** Multiplexer (MUX) for scaled addition.

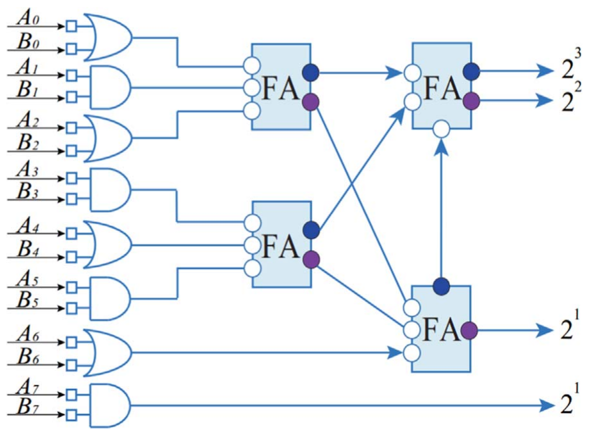

**Fig. 4.** Approximate parallel counters (APC) for stochastic addition.

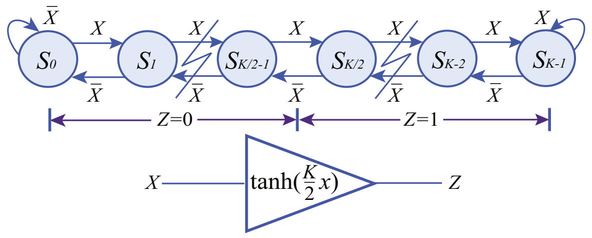

**Fig. 5.** Stochastic hyperbolic tangent.

### 4.1. APC-based inner product

Fig. 6 illustrates the APC-based hardware inner product design, where the multiplication is calculated using XNOR gates and addition is performed by an APC. 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 1 s in each column of the products. Note that the output of APC is a binary number with more than 1-bit width.

For a basic FEB design using APC for inner product, MUX for average pooling and the Btanh proposed in [20] for activation, the accuracy, area, power, and energy performance with respect to the input size are shown in Fig. 7 (a), (b), (c), and (d), respectively, under the fixed bit stream length 1024.

As illustrated in Fig. 7(a), a very slow accuracy degradation is observed as input size increases. However, the area, power, and energy of the entire FEB increases near linearly as the input size grows, as shown in Fig. 7(b),(c), and (d), respectively. The reason is as follows:

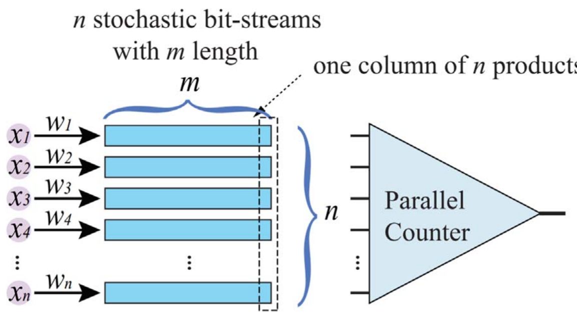

**Fig. 6.** APC-based inner product.

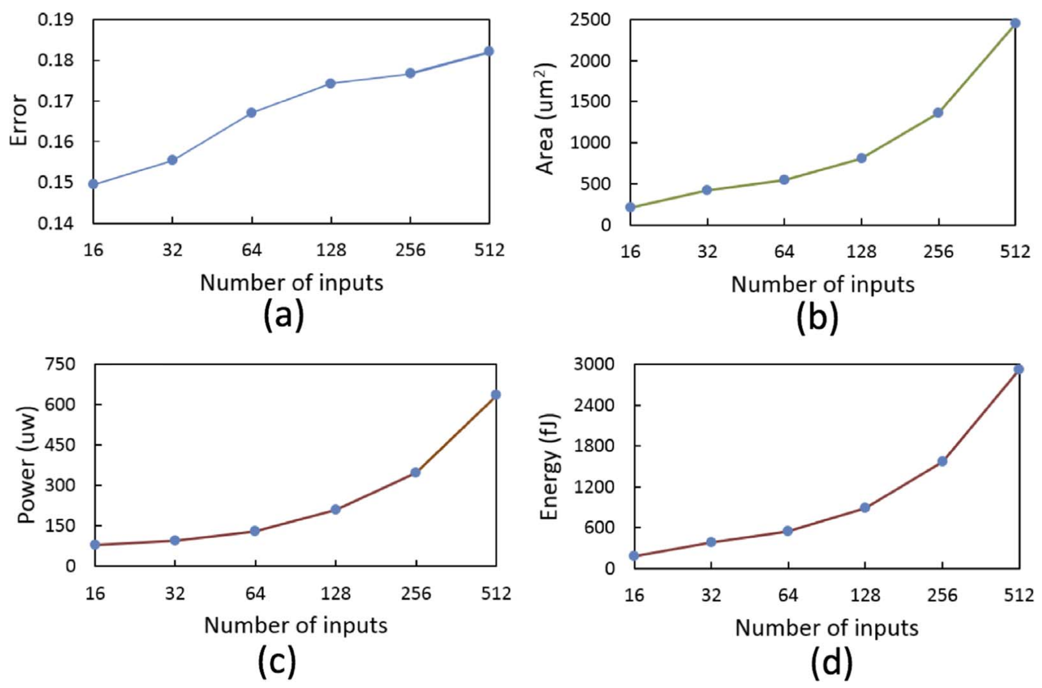

**Fig. 7.** Using the fixed bit stream length of 1024, the number of inputs versus (a) accuracy, (b) area, (c) power and (d) energy for an FEB using APC for inner product, MUX for average pooling and the Btanh proposed in [20] for activation.

With the efficient implementation of $Btanh(\cdot)$ function, the hardware 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 effective, we investigate the accuracy of FEBs using different stream lengths under different input sizes. As shown in Fig. 8, longer bit stream length consistently outperforms lower bit stream length in terms of accuracy in FEBs with different input sizes. However, designers should consider the latency and energy overhead caused by long bit streams.

As mentioned in Section 3.2.2, MUX can perform addition as well. Table 1 provides the performance between FEBs using APC-based inner product and MUX-based inner product under a fixed bit stream length equal to 1024 with different input sizes. Clearly, APC-based inner product is more accurate, more power efficient but has more area and latency than MUX-based inner product. As large-scale DCNN contains lots of FEBs with more than 64 inputs, the large absolute error in FEBs using MUX inner product will cause significant network performance degradation. Therefore, in this work, we only consider APC for inner product calculation.

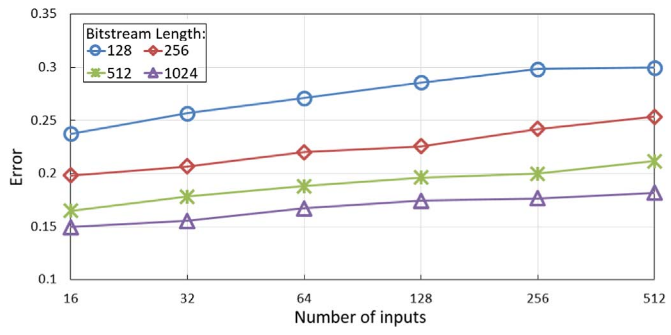

**Fig. 8.** The length of bit stream versus accuracy under different input numbers for an FEB using APC inner product, MUX based average pooling and Btanh activation.

## 4.2. Pooling design

Average pooling calculates a mean of a small matrix and can be implemented efficiently by a stack of MUXes. For four bit-streams representing pixels in a 2 × 2 region in a feature map, we can use a 4-to-1 MUX to calculate the mean of four bit-streams, as shown in Fig. 9 (a). Despite of the simple hardware implementation of average pooling, we consider max pooling in this work, which is adopted in most state-of-the-art DCNNs due to its better performance in practice. The authors in [15] proposed a hardware-oriented max pooling design, where the largest bit-stream in the most recent segment is selected as the near-max output, as shown in Fig. 9(b). Different from the hardware-oriented near-max pooling design in [15], we select the maximum among the current bit as output instead of predicting based on the most recent bits. Table 2 demonstrates the precision of the improved hardware-oriented near-max pooling for representative pooling units in DCNNs with up to 0.110 absolute error reduction compared with [15]. For a LeNet-5 [29] DCNN using MNIST dataset with 1024 bit stream, the network accuracy is improved by 0.11% using the improved hardware-oriented near-max pooling compared with the DCNN using max pooling in [15].

## 4.3. Activation design

The ReLU activation $f(x) = max(0, x)$ becomes the most popular activation function in state-of-the-art DCNNs [5]. This necessitates the design of SC-based ReLU block in order to accommodate the SC technique in state-of-the-art large-scale DCNNs, such as AlexNet for ImageNet applications. In this work, we adopt the SC-ReLU activation developed in [13]. Unlike tanh that generates output in the range of $[-1, 1]$, the ReLU always outputs non-negative number within $[0, 1]$. Accordingly, we use a shift register array to record the latest $\alpha$ bits of the output stochastic bit streams and a counter to calculate their sum. Hence, the SC-ReLU keeps tracking the last $\alpha$ output bits and predicts the sign of the current value based on the sum calculated by the counter. To be more specific, if the sum is less than half of the maximum achievable sum, the current value is predicted to be negative. Otherwise, it is predicted as positive (note that value 0 is half 1 s and half 0 s in the bipolar format). As the output cannot be negative for

**Table 1**
Comparison between APC-based FEB and MUX-based FEB using MUX for average pooling and tanh activation under 1024 bit stream.

| | FEB with APC Inner Product | | | FEB with MUX Inner Product | | | 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 ($\mu m^2$) | 209.9 | 417.6 | 543.2 | 110.7 | 175.3 | 279.8 | 189.7 | 238.2 | 194.1 |
| Power ($\mu W$) | 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 |

ReLU, the SC-ReLU activation mitigates such errors by outputting 1 (as compensation) whenever the predicted current value is negative.

## 5. Proposed normalization and dropout for SC-based DCNNs

### 5.1. Proposed stochastic normalization design

The overall stochastic normalization design is shown in Fig. 11. The structure follows the Local Response Normalization (LRN) equation presented in [1]. Given $a^i_{x,y}$ denotes the neuron computation results after applying kernel $i$ at position $(x, y)$ and ReLU nonlinearity activation function, output $b^i_{x,y}$ is calculated as

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2\right)^{\beta}} \tag{1}$$

where the summation part of this equation counts all the adjacent neuron outputs with $b^i_{x,y}$ and $N$ is the total number of neurons in this layer. $k$, $n$, $\alpha$ and $\beta$ are parameters that affect the overall accuracy of network. The validation set used in the AlexNet is $k = 2$, $n = 5$, $\alpha = 1$ and $\beta = 0.75$ [1].

The complex relationship in Eq.(1) is decoupled into three basic operations: (i) square and summation, which calculates $k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2$, (ii) activation, that performs the $(\cdot)^\beta$ operation, and (iii) division for the final output. Accordingly, the hardware structure for stochastic normalization design is separated into three units for the above-mentioned operations: (i) Square and Summation, (ii) Activation and (iii) Division.

#### 5.1.1. Square and summation

As show in Fig. 10, the stochastic square circuit consists of a D Flip-Flop and an XNOR gate. Squaring a signal in stochastic process is similar to multiplying 2 signals together. As mentioned in Section 3.2.1, multiplication is performed by XNOR gate. However, squaring a signal by using an XNOR gate alone will always result in a 1 since the two input signals in this case are correlated. To avoid this, a DFF is applied in this circuit to force one of the input signals to arrive late [28]. After delayed by one clock, the input signals become uncorrelated

**Table 2**
Precision of the improved max pooling for an FEB with 16-bit input size under 1024 bit stream.

| DCNN | Pooling | Segment size | Absolute error | Absolute error reduction over [15] |
|---|---|---|---|---|
| LeNet-5 [29] | 4-to-1 | 16 | 0.3126 | 0.110 |
| | | 32 | 0.2609 | |
| AlexNet [1] | 9-to-1 | 16 | 0.2143 | 0.095 |
| | | 32 | 0.2341 | |

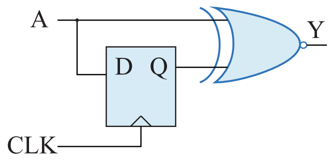

**Fig. 10.** Stochastic square circuit using a DFF and an XNOR gate.

and therefore an XNOR gate can be used to do multiplication. For the summation part, we use an APC mentioned in Section 3.2.2 to add up all the elements. Note that the output of the adder is in binary format.

#### 5.1.2. Activation function for normalization

FSM can be used to build stochastic approximation of activation functions. One challenge is that the max value that bipolar stochastic number can reach is 1 but the denominator of Eq.(1) can easily be greater than 1 in software normalization, e.g. $k > 1$ and $\beta > 1$. In order to resolve the above issue, we reshape the ReLU function (mentioned in Section 4.3) to imitate $(k + \alpha x)^\beta$. To be more specific, we change the slope and intercept of SC-ReLU activation to make its shape close to $(k + \alpha x)^\beta$ by re-configuring the hardware component of ReLU. During this process, the input range is set to $x \in [0, 1]$ and the output is limited to $y \in [0, 1]$. Here since $x \in [0, 1]$, we set $\alpha$ to be a constant 1. The imprecision of the activation operation can be compensated by jointly optimizing the parameters of activation unit and the following division unit, in order to make the final normalization result accurate.

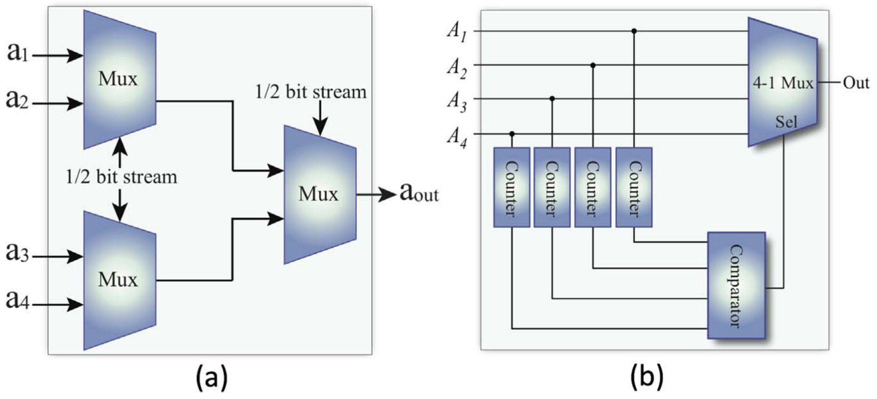

**Fig. 9.** Pooling design in SC: (a) average pooling and (b) near-max pooling.

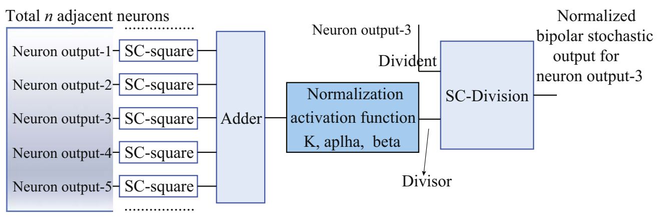

**Fig. 11.** The overall stochastic normalization design.

### 5.1.3. Division

Our Algorithm 1 outlines the proposed SC-Division design where steps 7–11 correspond to an AND operation between divisor input and the previous stochastic output. Only when both divisor input and previous feedback stochastic output are 1, the saturated counter is allowed to decrease. We have three different conditions to control the increment and decrement of the saturated counter as shown in steps 13–21. Besides, we also use a history shift register $H[0{:}\ \alpha-1]$ in order to count the last $\alpha$ bits of output stochastic bit stream. By using the sum of the $\alpha$ bits, which is denoted by $\delta$, we can predict the next stochastic output bit. Since the output is a non-negative bipolar stochastic number, we need to eliminate all possible negative error. To be more specific, if the sum $\delta$ is less than $\frac{\alpha}{2}$, the current value is predicted to be negative. Since each negative value raises an error, we mitigate such error by outputting a 1 instead of a 0.

**Table 3**
Absolute error of FEBs with 4-to-1 pooling (commonly used in LeNet-5 [29]) under different bit stream lengths and input sizes.

| Input size | Bit stream length | | | |
|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 |
| 16 | 0.154 | 0.152 | 0.152 | 0.151 |
| 32 | 0.115 | 0.112 | 0.110 | 0.112 |
| 64 | 0.086 | 0.084 | 0.084 | 0.080 |
| 128 | 0.071 | 0.066 | 0.064 | 0.066 |

**Algorithm 1.** Designated Division Algorithm for Normalization($m$, $x_j$, $y_j$, $\alpha$, $e$).

```
input : $m$ indicates bit stream length
input : $f$ is rate change for the FSM
input : $x_j$ is the input divisor bit of the $j$-th division block
input : $y_j$ is the input dividend bit of the $j$-th division block
input : $\alpha$ is the number of registers in the temporary array
input : $e$ is number of adjacent neurons
output: $z_k$ is the $k$-th stochastic output bit for the SC-division
1  $S_{max} \leftarrow e-1$;                                  /* max state */
2  $S \leftarrow \frac{e}{2}$;                                 /* current state */
3  $r \leftarrow \alpha-1$;                                    /* r is an iterator */
4  $H[0:\alpha-1] \leftarrow 0$;                               /* initialize history array */
5  $\delta \leftarrow 0$;                                      /* initialize shadow counter */
6  for $k \leftarrow 1$ to $m$ do
7      if $S > \frac{e}{2}$ then                               /* Divisor */
8          $X \leftarrow x_j$
9      else
10         $X \leftarrow 0$
11     end
12     $Y = y_j$;                                              /* Dividend */
13     if $X == 1$ then                                        /* Next state logic */
14         $S = S - f$
15     end
16     if $X == 0 \&\& Y == 1$ then
17         $S = S + 2f$
18     end
19     if $X == 0 \&\& Y == 0$ then
20         $S = S$
21     end
22     if $S < 0$ then                                         /* saturated counter */
23         $S \leftarrow 0$
24     else if $S > S_{max}$ then
25         $S \leftarrow S_{max}$
26     end
27     if $\delta < \frac{\alpha}{2}$ then                      /* compensate for negative value */
28         $z_k^j \leftarrow 1$
29     else
30         else if $S > \frac{e}{2}$ then
31             $z_k^j \leftarrow 1$
32         else
33             $z_k^j \leftarrow 0$
34         end
35     end
36 end
37 while $r \geq 1$ do
38     $H[r] \leftarrow H[r-1]$;                               /* update the history array */
39     $r \leftarrow r-1$
40 end
41 $H[0] \leftarrow z_k$
42 $\delta \leftarrow \sum_{r=0}^{\alpha-1} H[r]$;              /* update the shadow counter */
43 $r \leftarrow \alpha-1$
```

**Table 4**
Absolute error of FEBs with 9-to-1 pooling (commonly used in AlexNet [1]) under different bit stream lengths and input sizes.

| Input size | Bit stream length | | | |
|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 |
| 16 | 0.120 | 0.121 | 0.117 | 0.119 |
| 32 | 0.073 | 0.064 | 0.065 | 0.065 |
| 64 | 0.051 | 0.037 | 0.031 | 0.033 |
| 128 | 0.042 | 0.026 | 0.021 | 0.018 |

### 5.2. Integrating dropout into SC-Based DCNN

DCNN can automatically learn complex functions directly from raw data by extracting representations at multiple level of abstraction [5]. The deep architecture, together with advanced training algorithms, has significantly enhanced the self-learning capacity of DCNN. However, the learning process to determine the parameters in DCNNs becomes computationally intensive due to the large number of layers and parameters. The dropout technique is proposed to randomly omit feature detectors with a certain probability $p$ (usually $p = 0.5$) on each training case [1]. After applying dropout, a hidden neuron cannot rely on the particular other hidden neurons presence and has to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. Accordingly, complex co-adaption of neurons on the training data is prevented. Applying dropout can also be viewed as training lots of different networks and averaging their predictions. The trained weights need to be multiplied by $p$ as an approximation to taking the geometric mean of the prediction distributions produced by the dropout networks. The dropout technique is applied in both training and inference. The training time is shortened due to the simplified network for each input training data, whereas the performance is improved as overfitting is mitigated. Note that there is no hardware overhead for applying dropout since only learned weights need to adjusted for the SC-based DCNN.

## 6. Experimental results

In this section, we present (i) performance evaluation of the SC FEBs, (ii) performance evaluation of the proposed SC-LRN design, and (iii) impact of SC-LRN and dropout on the overall DCNN performance. The FEBs and DCNNs are synthesized in Synopsys Design Compiler with the 45 nm Nangate Library [30] using Verilog.

### 6.1. Performance evaluation of the proposed FEB designs

For the overall FEB Designs, the accuracy depends on the bitstream length and input size. In order to see the effects of the aforementioned factors, we analyze the inaccuracy based on a wide range of input sizes and bitstream lengths for FEBs using 4-to-1 pooling (used in LeNet-5 [29]) and 9-to-1 pooling (deployed in AlexNet [1]), as shown in Table 3 and Table 4, respectively. Noted that the inaccuracy here is calculated compare to software results. One can observe from Tables 3 and 4 that increasing the bit stream length tends to reduce the absolute error of FEB under a given input size. Besides, with the improved near-max pooling design, the FEB tends to be more accurate as input size increases under a fixed bit stream length. This means that for larger neurons deployed in large-scale DCNNs (i.e., FEB with larger input size), the proposed FEB will be more accurate.

The corresponding hardware area, power and energy of proposed FEB (with 4-to-1 pooling) are shown in Fig. 12 (a), (b) and (c), respectively. As for the 9-to-1 pooling FEB, the hardware area, power and energy of proposed FEB are shown in Fig. 13 (a), (b) and (c), respectively. Despite of the reduced absolute error, the area, power, and energy for both FEBs all increase as the input size increases. Note that max pooling is placed before ReLU in the FEB with 4-to-1 pooling,

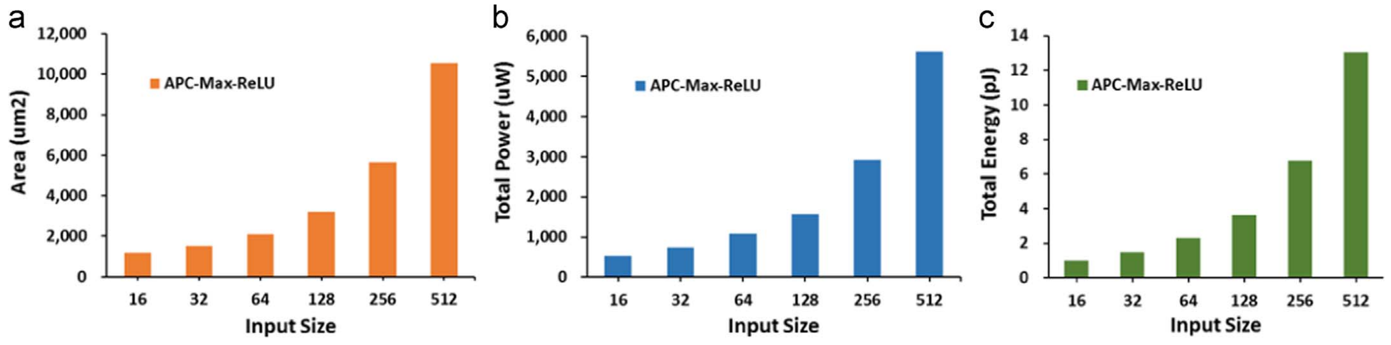

**Fig. 12.** Input size versus (a) total power, (b) area, and (c) total energy for the FEB design (4-to-1 pooling).

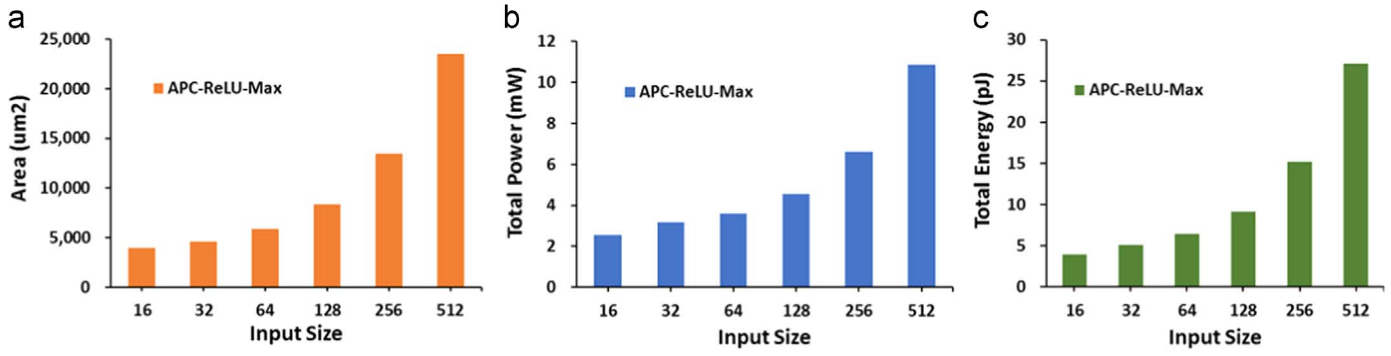

**Fig. 13.** Input size versus (a) total power, (b) area, and (c) total energy for the FEB design (9-to-1 pooling).

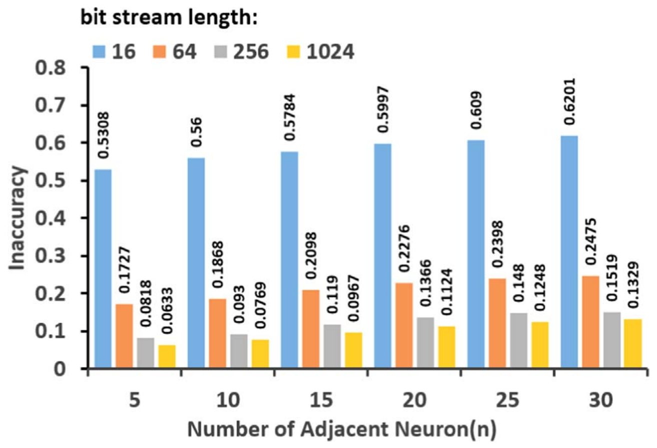

**Fig. 14.** Number of adjacent neurons versus absolute inaccuracy under different bit stream lengths for proposed LRN.

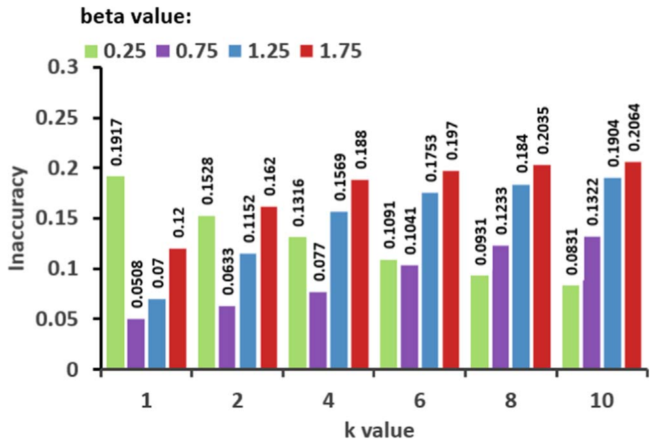

**Fig. 15.** Different K values versus absolute inaccuracy under different $\beta$ for the proposed LRN.

whereas pooling in after ReLU in the FEB with 9-to-1 pooling. This order of pooling and ReLU operation follows the software implementation in the original LeNet-5 [29] and AlexNet [1].

## 6.2. Performance evaluation of the proposed SC-LRN designs

As for SC-LRN, several parameters may result in accuracy degradation, i.e., $n$, $\beta$, $k$ and bit stream length. The set of parameters used in software cannot be directly applied here due to the imprecision induced by the stochastic components. Hence, we need to run experiments to find the best setting of these parameters in the proposed SC-LRN.

We set up the first experiment between the inaccuracy of SC-LRN under different bit stream length and number of adjacent neurons, and the results are illustrated in Fig. 14. It is obviously that, as $n$ increase, there is a continuous accuracy decrement regardless of bit stream length. In addition, longer bit stream length yields higher accuracy, and this indicates that the precision can be leveraged by adjusting the bit stream length without hardware modification.

In the second experiment, we evaluate the effect of $\beta$ and $k$ to the overall accuracy of SC-LRN under a wide range of $\beta \in [0, 2]$ and $k \in [1, 12]$, and the results are provided in Fig. 15. Due to the fact that $(k + \alpha x)^{\beta}$ is not a monotone function, we need to discuss the inaccuracy based on the range of $k$ and $\beta$. When $\beta > 0.75$, increasing $k$ value will result in imprecision. If $\beta < 0.25$, the accuracy will increase as $k$ increase. For $0.25 \le \beta \le 0.75$, the accuracy of SC-LRN is barely affected by $k$.

In the third experiment, we further compare the proposed SC-based LRN with the binary ASIC hardware LRN design. The parameter values we choose for the tests are as follow: n = 5, k = 2 and $\beta$ = 0.75. Clearly, the number of bits in fixed-point number affect both the hardware cost and accuracy. To make the comparison fair, we adopt minimum fixed point (8 bit) that yields a DCNN network accuracy that is almost identical to the software DCNN. Table 5 shows the performance comparison between binary-LRN and SC-LRN. Compared with binary ASIC LRN, the proposed SC-LRN achieves up to 5×, 9×, and 6× improvement in terms of power, energy and area, respectively, indicating significant hardware savings with only a little accuracy deficit.

**Table 5**
SC-LRN versus Binary-LRN hardware cost.

| Type | Area(μm$^2$) | Power(μW) | Energy(fJ) | Delay(ns) | Inaccuracy |
|---|---|---|---|---|---|
| SC-LRN | 290.2 | 64.6 | 77.5 | 1.2 | 0.06 |
| Binary-LRN | 1786.7 | 333.5 | 700.4 | 2.1 | 0.04 |

**Table 6**
AlexNet accuracy results.

| No. | Configuration | Model Accuracy | | Inference Accuracy | |
|---|---|---|---|---|---|
| | | Top-1 (%) | Top-5 (%) | Top-1 (%) | Top-5 (%) |
| 1 | Original with LRN & Dropout [1] | 57.63 | 81.35 | 56.49 | 80.47 |
| 2 | w/o Dropout | 55.83 | 79.94 | 54.75 | 78.86 |
| 3 | w/o LRN | 55.97 | 80.53 | 54.94 | 79.73 |
| 4 | w/o Dropout & LRN | 54.25 | 78.37 | 53.23 | 77.42 |

## 6.3. Impact of SC-LRN and dropout on the overall DCNN performance

In this section, we re-train AlexNet [1] models on ImageNet challenge [31] with four different configurations, (1) Original AlexNet (with both LRN and Dropout), (2) AlexNet without Dropout, (3) AlexNet without LRN, and (4) AlexNet without Dropout and LRN. Please note we do not pre-process the ImageNet data with data augmentation as [1] suggested and we scale the input image pixel values to [0,1] from [0,255] so that the inputs fed into the network range from [−1,1] after processing. Different software-based model accuracies on the test set are achieved. Then we evaluate the SC-based inference accuracy with SC-based components (including proposed LRN and Dropout) given bit-stream length as 1024 to show the application-level degradation from trained models.

As shown in Table 6, we observe that with the proposed SC-LRN and dropout, the hardware inference accuracy can achieve top-1 and top-5 accuracy as high as 57.63% and 81.35%, respectively. Please note that top-1 accuracy counts when the predicted label with the highest probability is exactly the same as the ground-truth while top-5 accuracy counts when the ground-truth falls in the first five predicted labels with highest probabilities. The top-1 and top-5 accuracy degradations of hardware based designs are about 1%, which is a small degradation from the software DCNNs. Furthermore, we can see from No. 2 and No. 4 that the network degrades by only 0.1–0.2% with SC-LRN; from No. 3 and No. 4, Dropout nearly shows no degradation compared with software trained model accuracies. Finally, the No. 1 configuration with the proposed SC-LRN and Dropout achieves 3.26% top-1 accuracy improvement and 3.05% top-5 accuracy improvement compared with the No. 4 configuration without these two essential techniques, demonstrating the effectiveness of proposed normalization and dropout designs.

## 7. Conclusion

We presented hardware implementations of normalization and dropout for SC-based DCNNs. First, FEBs in DCNNs were built with APC for inner product, an improved near-max pooling block for pooling, and SC-ReLU for activation. Then, a novel SC-based LRN design was proposed, comprising square and summation unit, activation, and division units. The dropout technique was integrated in the training phase and the corresponding weights were adjusted for the hardware implementation. Experimental results on AlexNet validated the effectiveness of the proposed LRN and dropout design.

## References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. (2012) 1097–1105.

[2] B. Hu, Z. Lu, H. Li, Q. Chen, Convolutional neural network architectures for matching natural language sentences, Adv. Neural Inf. Process. Syst. (2014) 2042–2050.

[3] K. Simonyan, A. Zisserman, Two-stream convolutional networks for action recognition in videos, Adv. Neural Inf. Process. Syst. (2014) 568–576.

[4] T.N. Sainath, A.R. Mohamed, B. Kingsbury, B. Ramabhadran, Deep convolutional neural networks for lvcsr, in: Proceedings of the 2013 IEEE international conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2013, pp. 8614–8618.

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

[6] Y. Xue, J. Li, S. Nazarian, P. Bogdan, Fundamental challenges toward making the iot a reachable reality: a model-centric investigation, ACM Trans. Des. Autom. Electron. Syst. (TODAES) 22 (3) (2017) 53.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM international conference on Multimedia, ACM, 2014, pp. 675–678.

[8] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron, et al., Theano: deep learning on gpus with python, in: NIPS 2011, BigLearning Workshop, Granada, Spain, vol. 3, Citeseer, 2011.

[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, 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, ACM, 2015, pp. 161–170.

[10] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al., Dadiannao: a machine-learning supercomputer, in: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, 2014, pp. 609–622.

[11] Y.-H. Chen, T. Krishna, J. S. Emer, V. Sze, Eyeriss: An Energy-efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks, IEEE Journal of Solid-state Circuits.

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

[13] J. Li, Z. Yuan, Z. Li, C. Ding, A. Ren, Q. Qiu, J. Draper, Y. Wang, HaRdware-driven Nonlinear Activation for Stochastic Computing Based Deep Convolutional Neural Networks, Arxiv Preprint http://arXiv.org/abs/arXiv:1703.04135arXiv:1703.04135.

[14] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, B. Yuan, Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks, in: Proceedings of the IEEE 34th International Conference on Computer Design (ICCD), 2016, pp. 678–681.

[15] A. Ren, J. Li, Z. Li, C. Ding, X. Qian, Q. Qiu, B. Yuan, Y. Wang, Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing, in: Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, 2017, pp. 405–418.

[16] Z. Li, A. Ren, J. Li, Q. Qiu, B. Yuan, J. Draper, Y. Wang, Structural design optimization for deep convolutional neural networks using stochastic computing, in: Proccedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2017, pp. 250–253.

[17] Z. Yuan, J. Li, Z. Li, C. Ding, A. Ren, B. Yuan, Q. Qiu, J. Draper, Y. Wang, Softmax regression design for stochastic computing based deep convolutional neural networks, in: Proceedings of the on Great Lakes Symposium on VLSI 2017, ACM, 2017, pp. 467–470.

[18] Y. Ji, F. Ran, C. Ma, 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.

[19] J. Li, J. Draper, Accelerated soft-error-rate (ser) estimation for combinational and sequential circuits, ACM Trans. Des. Autom. Electron. Syst. (TODAES) 22 (3) (2017) 57.

[20] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, K. Choi, 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.

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

[22] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, arXiv preprint ⟨arXiv:http://arXiv.org/abs/arXiv:1207.0580arXiv:1207.0580⟩.

[23] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.

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

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

[26] C. Zhang, Z. Fang, P. Zhou, P. Pan, J. Cong, Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks, in: Proceedings of the 35th International Conference on Computer-Aided Design, ACM, 2016, p. 12.

[27] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, W.J. Dally, Eie: efficient inference engine on compressed deep neural network, in: Proceedings of the 43rd International Symposium on Computer Architecture, IEEE Press, 2016, pp. 243–254.

[28] B.D. Brown, H.C. Card, Stochastic neural computation. I. Computational elements, IEEE Trans. Comput. 50 (9) (2001) 891–905.

[29] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al., Comparison of learning algorithms for handwritten digit recognition, in: Proceedings of the International Conference On Artificial Neural Networks, Networks, vol. 60, Perth, Australia, 1995, pp. 53–60.

[30] Nangate 45 nm Open Library, Nangate Inc., 2009.[link] URL ⟨http://www.nangate.com/⟩.

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: a large-scale hierarchical image database, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, IEEE, 2009, pp. 248–255.