TELKOMNIKA, Vol.16, No.6, December 2018, pp.2835~2843
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v16i6.8955

Stochastic Computing Correlation Utilization in Convolutional Neural Network Basic Functions

Hamdan Abdellatef*, Mohamed Khalil-Hani, Nasir Shaikh-Husin, Sayed Omid Ayat
VeCAD Research Laboratory, School of Electrical Engineering, Faculty of Engineering,
Universiti Teknologi Malaysia, Johor Bahru, Johor, Malaysia
*Corresponding author, e-mail: hamdan.abdellatef@gmail.com

Abstract

In recent years, many applications have been implemented in embedded systems and mobile Internet of Things (IoT) devices that typically have constrained resources, smaller power budget, and exhibit "smartness" or intelligence. To implement computation-intensive and resource-hungry Convolutional Neural Network (CNN) in this class of devices, many research groups have developed specialized parallel accelerators using Graphical Processing Units (GPU), Field-Programmable Gate Arrays (FPGA), or Application-Specific Integrated Circuits (ASIC). An alternative computing paradigm called Stochastic Computing (SC) can implement CNN with low hardware footprint and power consumption. To enable building more efficient SC CNN, this work incorporates the CNN basic functions in SC that exploit correlation, share Random Number Generators (RNG), and is more robust to rounding error. Experimental results show our proposed solution provides significant savings in hardware footprint and increased accuracy for the SC CNN basic functions circuits compared to previous work.

Keywords: *convolutional neural network, stochastic computing, correlation*

1. Introduction

Deep 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. Convolutional Neural Network (CNN) is recognized as one of the most promising types of Artificial Neural Networks (ANN) and has become the dominant approach for almost all recognition and detection tasks [1] such as face recognition [2], handwritten digit recognition [3], target recognition [4], and image classification [5]. To achieve acceptable classification results, CNN performs a massive number of convolutions and sub-sampling operations with significant amounts of intermediate data results. Despite its high classification accuracy, a deep CNN is highly demanding in terms of energy consumption and computation cost [6]. To bring the success of CNNs to resource-constrained mobile and embedded systems, designers must overcome the challenges of implementing resource-hungry CNNs in embedded systems with limited area and power budget [7].

Stochastic Computing (SC), which represents and processes information in the form of a probability of ones in a bit-stream, has the potential to implement CNNs with significantly reduced hardware resources and achieve high power efficiency. In SC, arithmetic operations like multiplication can be performed using simple AND or XNOR logic gate in Uni-Polar (UP) or BiPolar (BP) representation, respectively, and scaled addition is done using Multiplexers (MUX). Also, in SC, there are no positional weights among bits; therefore, SC circuits are better in soft-error resilience and have a free dynamic trade-off between performance, accuracy, and energy. Fortunately, neural networks have high error-tolerance at the algorithmic level, which allows using SC for CNN implementation [8]. Opposite to what was previously believed, the correlation among Stochastic Numbers (SN) can serve as a resource in designing stochastic circuits. A comparative study [9] reported that the circuits exploiting correlation are generally smaller, more accurate, and have lower latency than those with independent inputs. However, previous SC CNN works [7, 8, 10, 11] did not explore correlation in their implementation. Besides, these works did not consider the conversion circuits between SC and conventional binary in their estimates as RNGs can take up to 80% of the circuit cost [12]. Also, long bit

Received February 14, 2018; Revised September 27, 2018; Accepted October 24, 2018

streams up to 8192 bits are used to obtain acceptable accuracy which significantly increases the latency. This work has the following contributions:

1. SC CNN basic functions (inner product, pooling, and ReLU activation function) that exploit correlation is proposed. The obtained functions had significant lower resource utilization, higher accuracy, and more robust to rounding error compared to previous SC work. Also, it shows significant area reduction compared to binary implementation [13].
2. A new method that generates uncorrelated bit streams for MUXs selectors to enhance the accuracy of scaled addition using Toggle Flip-Flops (TFF) is presented. Also, this method reduces the number of used RNGs.

The rest of this paper is organized as follows. Section 2 overviews CNN and explains its basic functions. Section 3 reviews SC basics. Section 4 presents the CNN basic functions (inner product, pooling, ReLU activation function) design method. Section 5 presents experimental results to show the effectiveness of the proposed design with respect to compactness and accuracy, and finally, Section 6 concludes the paper.

2. Convolutional Neural Network

Previously, hand-engineered features development had been the primary source of difficulty in computer vision, like sophisticated feature extractors, to identify higher-level patterns that are optimal for machine vision tasks, such as object recognition. However, convolutional neural networks aim to solve this problem by learning higher-level representations automatically from data [14]. As a supervised learning algorithm, CNN employs a feedforward process for recognition and a backward path for training. In industrial practice, many application designers train CNN off-line and use the off-line trained CNN to perform time-sensitive jobs. Thus, the speed, area, and energy consumption of feedforward computation are to be considered. This work is scoped to the feedforward computation hardware implementation on FPGA.

A typical CNN, as shown in Figure 1, is composed of multiple computational layers that can be categorized into two components: a feature extractor and a classifier. The feature extractor is used to filter input images into "feature maps" that represent features of the image such as corners and edges which are relatively invariant to position shifts or distortions. The feature extractor output is a low-dimensional vector containing features. Then, the vector is fed into the CNN second component classifier, which is usually a traditional fully connected artificial neural network. The classifier decides the likelihood of categories that the input image might belong to [15].

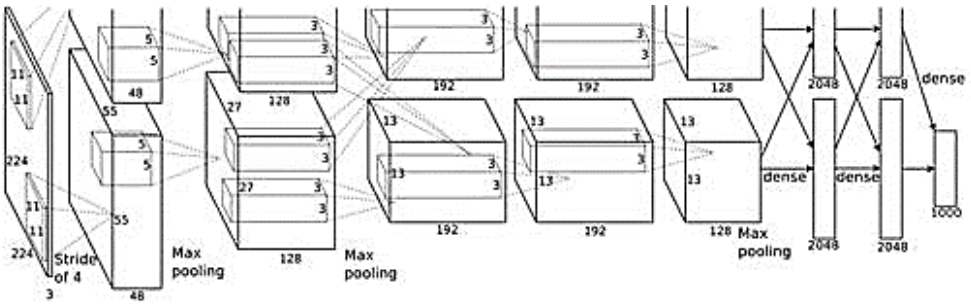


Figure 1. CNN architecture proposed in [13]

The CNN layers can be one of three types: convolutional layer, pooling layer, or fully connected layer. The convolutional layer is the core building block of the CNN whereas the main operation is the convolution that computes the inner-product of receptive fields, a window of the input feature map, and a set of learnable filters. This layer is the most time and resource consuming operation in CNN, occupying more than 90% overall computation dominating runtime and energy consumption as shown in [16]. The convolutional layer has N feature map batch size, M output feature maps, Ch input feature maps, H/W size of output feature map, K weights kernel size, and S stride. To compute one element in the output feature map of the

convolutional layer (1) is evaluated. This layer is of 4-D operation and to calculate all output values (1) is looped for N, M, H, and W . The convolutional layer contains two functions, the convolution, and activation. The convolution is loops of multiply-accumulate (MAC) operations (or inner product). The activation functions conduct non-linear transformations such as Rectified Linear Unit (ReLU), Sigmoid function, and hyperbolic tangent function. This work implements only ReLU activation function since it is the most popular one. ReLU compares the input to zero and outputs the maximum value as shown in (2).

$$O[n][m][r][c] = Activation(B[m] + \sum_{ch=0}^{Ch-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} x[n][ch][Sr+i][Sc+j] \times w[m][ch][i][j] \quad (1)$$

$$out = max(x, 0) \quad (2)$$

The pooling layers perform nonlinear down-sampling for data dimension reduction. Commonly, max pooling and average pooling are used for this purpose. The max pooling layer is shown in (3) which output the max value in a 2D window of K size and S stride. The average pooling is to compute the average value of the same window as shown in (4). To complete the layer operation, this equation is repeated for all N, M, H, W. The output feature map of the pooling layer has a $1/S$ dimension reduction in height and width. Finally, the high-level reasoning is completed via the classifier which is a fully connected layer. Neurons in this layer are connected to all activation results in the previous layer which is an inner product with a filter size of one element.

$$O[n][m][r][c] = max(x[n][m][Sr+i][Sc+j]) \ \ for\ (0,0) \leq (i,j) < (K-1, K-1) \quad (3)$$

$$O[n][m][r][c] = \frac{1}{K^2} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} x[n][m][Sr+i][Sc+j] \quad (4)$$

The basic operations in CNN are the inner product, pooling, and activation function operations. Any CNN neuron may consist of one or multiple basic operations. For instance, neurons in convolutional layers implement inner product and activation operations only; those in pooling layers implement pooling only, and those in fully connected layers implement inner product and activation operations.

3. Stochastic Computing

Stochastic computing (SC) represents and processes information in the form of digitized probabilities. In SC, numbers called stochastic numbers SNs are represented by binary bit streams. The SN donate the probability p, the probability of 1s in the SN [17]. SN has no fixed length nor structure. The stochastic representation can be One-line UP, One-line BP, and Two-line. This paper uses BP representation since it allows negative values, but UP do not. The UP and BP stochastic representations are clarified in Table 1 where N_0, N_1, and N represents the number of zeros, ones, total bits in SN respectively. To convert from binary to stochastic a stochastic number generator (SNG) is used which is a random number generator RNG and a comparator. On the other hand, to convert from Stochastic to binary, a counter is used. According to [18], SNs should be independent and uncorrelated bit-streams. However, recent studies [9] showed that the correlation could serve as a resource in designing stochastic circuits. In that study, Alaghi and Hayes introduced a parameter that determines the significance of the correlation between two SNs called stochastic computing correlation (SCC) as shown in (5). The major advantage of SC is that it employs very low-complexity arithmetic units [17], as shown in Table 1. The AND or XNOR gates perform the multiplication operation in SC using UP or BP representation. There is no direct addition in SC; instead, a scaled addition is used. 2-to-1 MUX performs the scaled addition having $p_s = 0.5$ or $s = 0$ selector bit stream value in UP and BP representation respectively. It should be noted that the MUX selector should be uncorrelated with the MUX inputs to prevent correlation-induced error. However, the MUX inputs are correlation-insensitive and can have any value of correlation. To perform subtraction NOT gate can be used to negate the subtracted value and then it will be added by 2-to-1 MUX to the other value.

Table 1. The SN Representations and Basic Operations

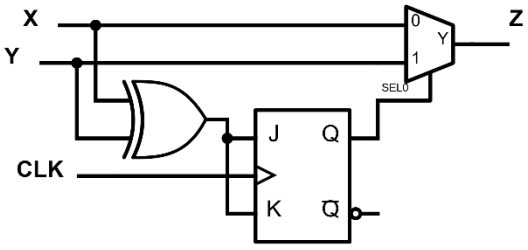
	Value	Interval	Relation	Negation	Multiplication	Scaled Addition
UP	$p = \frac{N_1}{N}$	[0,1]	$\mathrm{p} = \frac{1+x}{2}$	$p_{NOT} = 1 - p$	$p_{AND} = p_1 p_2$	$p_{MUX} = p_s p_1 + (1 - p_s) p_2$
BP	$x = \frac{N_1 - N_0}{N}$	[-1,1]	$\mathrm{x} = 2\mathrm{p} - 1$	$x_{NOT} = -x$	$x_{XNOR} = x_1 x_2$	$x_{MUX} = \frac{1}{2}[(1+s)x_1 + (1 - s)x_2]$

One source of inaccuracy in SC is rounding. If we wish to eliminate the rounding error, the SN length L, and the binary precision, the number of bits, n should satisfy $\mathrm{L} = 2^n$. Suppose the precision in binary computing is $\mathrm{n} = 8$ bits, so the full SN length $\mathrm{L} = 256$ bits. Each bit requires one clock cycle to be processed causing the SC latency. As precision increases, L and latency increase exponentially which is a significant drawback in SC. To reduce the number of clock cycles needed, the full SN length is not used producing rounding error.

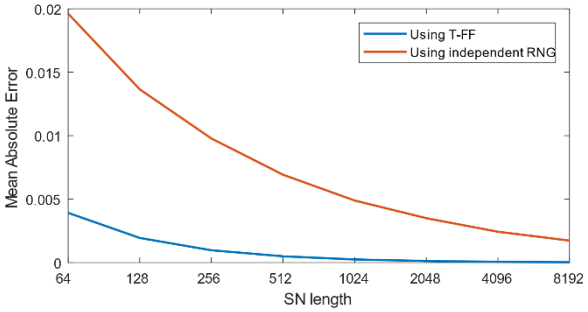
$$SCC(X,Y) = \begin{cases} \frac{p_{X\cap Y} - p_X p_Y}{min(p_X, p_Y) - p_X p_Y} & p_{X\cap Y} - p_X p_Y > 0 \\ 0 & p_{X\cap Y} - p_X p_Y = 0 \\ \frac{p_{X\cap Y} - p_X p_Y}{p_X p_Y - max(p_X + p_Y - 1, 0)} & p_{X\cap Y} - p_X p_Y < 0 \end{cases} \quad (5)$$

For SC addition operation, it is required to produce a 0.5 bit-stream that has $SCC \approx 0$ with respect to the inputs. Theoretically, the independent RNGs generates uncorrelated bitstreams, but the growing circuit size will require many independent RNGs affecting the area cost. In this study, we propose using flip-flops (FF)s to obtain uncorrelated bit-streams for SC scaled addition. T-FF is a JK-FF where T is connected to both inputs of the JK-FF. Based on Gaudet and Rapley [19] the JK-FF output follow (6). In the case of T-FF $p_T = p_J = p_K$, then always $p_Q = 0.5$ for any value of T and always $\mathrm{SCC(T, Q)} \approx 0$. Using T-FF will allow using one RNG for all SNGs to generate the MUX inputs and the uncorrelated selector as shown in Figure 2a. Similarly, SC addition accuracy will be increased as shown in Figure 2b.

$$p_Q = \frac{p_J}{p_J + p_Q} \quad (6)$$



(a) SC scaled addition using T-FF for MUX selector generation



(b) Accuracy comparison between T-FF and independent RNG for selector generation

Figure 2. More accurate SC scaled addition by using T-FF

4. Design of Stochastic Computing Based Convolutional Neural Network Basic Functions

4.1. Inner Product

The inner product is a MAC operation which is the basic function of convolution in CNN. The number of elements in the inner product is determined from the "for loop" unroll factor. To perform inner product in SC, the standard blocks are XNOR to perform multiplication and MUX or Approximate Parallel Counter (APC) [20] to perform addition as proposed in previous SC

CNN works [7, 8, 10]. However, XNOR gate requires long uncorrelated bit-streams which will affect latency and area cost by using one RNG per SNG. On the other hand, MUX tree approach proposed in [21] to perform inner product in digital filter case study can be adapted to this application. MUX tree allows sharing RNG among all inputs and is more accurate compared to previous XNOR-MUX and XNOR-APC approaches as to be shown in Section 5.

One MUX can be used for inner product of 2-elements vectors x and h as shown in Figure 3 and following (7), where the real operation does not involve multiplication or

$$z = \frac{1}{\sum|h|}(h \cdot x) = \frac{1}{|h_1|+|h_2|}(h_1x_1 + h_2x_2) \tag{7}$$

addition. The selector bit-stream of probability $\frac{|h_1|}{|h_1|+|h_2|}$ which follows (8) and

$$sl_M = \frac{\sum(h1_M)}{\sum(h1_M \cup h0_M)} \tag{8}$$

$\text{sign}(h_i)$ will be denoted by s_i. The same equation shows the MUX tree output for inner product of two vectors x and h with any length, but scaling seems to be a problem. Taking advantage of learning, the backward phase of the network is modified to adapt the scaling.

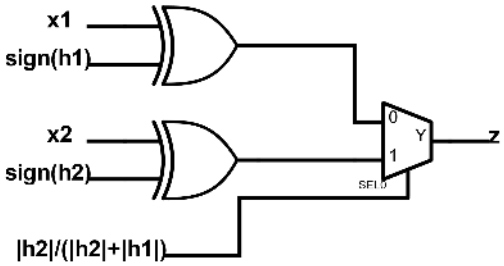


Figure 3. SC inner product circuit [21]

To create a mathematical model for SC inner product using the MUX tree and adapt it to CNN, the MUX tree will be changed to the sum of products. For N_{in} inputs, the MUX tree has $N_{in} - 1$ MUXs. We define the SM array which is a multi-dimensional array created from the selector probabilities. This array will be of dimensions N_{in}. Each element is a binary ANDing of selector bits related to the specific input until the output (9) shows the general SM array and an example of OL-MUX tree if $N_{in} = 5$ where sl_i is a bit of the selector bit-stream of the i^{th} MUX. For more information about constructing optimum OL-MUX trees we refer to [21].

$$SM = \begin{bmatrix} \overline{sl_{N_{in}-1}} \cap & \cdots & \cap \overline{sl_1} \\ \overline{sl_{N_{in}-1}} \cap & \cdots & \cap sl_1 \\ \vdots & \ddots & \vdots \\ sl_{N_{in}-1} \cap & \cdots & \end{bmatrix} = \begin{bmatrix} \overline{sl_4} \cap \overline{sl_2} \cap \overline{sl_1} \\ \overline{sl_4} \cap \overline{sl_2} \cap sl_1 \\ \overline{sl_4} \cap sl_2 \\ sl_4 \cap \overline{sl_3} \\ sl_4 \cap sl_3 \end{bmatrix} \tag{9}$$

Thus, the general equation of the MUX tree will become like that of (10) for evaluating one bit of the output bit-stream.

$$Z = \cup_{i=0}^{N_{in}-1}(s_i \oplus x_i) \cap SM_i \tag{10}$$

Suppose we want to use the SC inner product MUX tree to compute all elements (e.g. $N_{in} = Ch \times K \times K$). The inner product in (1) can be changed to (11) by taking advantage of SC (7) and (10) and adding a new loop representing the SN bits. It should be noted that all the

variables are bits. The new index [b] is to evaluate the bit operation through the SNs from 0 to L – 1. It is apparent the amount of resource utilization reduced.

$$O[n][m][r][c][b] = \cup_{ch=0}^{Ch-1} \cup_{i=0}^{K-1} \cup_{j=0}^{K-1}((s_{w[m][ch][i][j]} \oplus x[n][ch][Sr+i][Sc+j][b]) \cap SM[m][ch][i][j][b] \quad (11)$$

Instead of using a multiplier and an adder, by SC we use only some simple gates at the expense of latency. The resources used in a MUX tree inner product of N_{in} number of parallel inputs are N_{in} XOR gates and $N_{in} - 1$ MUXs. In practical implementations, to highly reduce latency, some loops should be unrolled entirely.

The inputs of CNN are of single size. To use the full SN length, L should be $L = 2^{32} = 4294967296$ which is too long and will produce high latency. One approach to reduce L is to reduce n, binary precision that will cause the binary quantization. The other approach is to reduce L without changing n which will cause the SC rounding error. The accuracy of MUX tree inner product circuit is evaluated with respect to precision and number of inputs with rounding error as shown in Table 2. It can be concluded that the MUX tree has high accuracy and robust to rounding error.

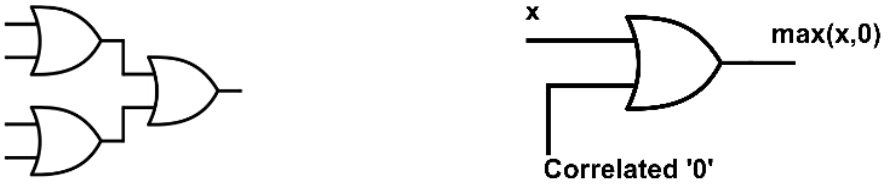
Table 2. MUX Tree Absolute Error $\times 10^{-2}$ with Respect to SN Length L and Number of Inputs, N_{in}, using Binary Precision $n = 64$

N_{in}, L	64	128	256	512	1024	2048	4096	8192
2	3.3	2.2	1.49	1.02	0.73	0.51	0.36	0.25
4	4.48	3.07	2.13	1.48	1.05	0.74	0.53	0.37
8	5.03	3.6	2.47	1.74	1.22	0.86	0.62	0.42
16	5.45	3.81	2.66	1.85	1.32	0.94	0.66	0.47

4.2. Pooling and Activation Function

In SC, if the correlation is exploited, the OR gates act as the max function for SCC=1. Therefore, (3) can be modified to become as stated in (12). In SC, instead of using a compactor, the OR gate will perform the max operation leading to a significant reduction in hardware footprint. Usually, the max pooling stride S is 2 and kernel K is 2, so max pooling can be realized using only 3 OR gates using the proposed approach after unrolling i and j loops of (12) as shown in Figure 4a. Similar to max pooling, the ReLU activation function performs the max operation but compared to zero. Thus, the input is "ORed" with a correlated SN of x=0 in the BP domain. Figure 4b shows the ReLU circuit.

$$O[n][m][r][c][b] = \cup_{i=0}^{K-1} \cup_{j=0}^{K-1}(X[n][m][Sr+i][Sc+j][b] \quad (12)$$



(a) Proposed max pooling

(b) Proposed SC ReLU circuit

Figure 4. Proposed SC CNN pooling and activation function circuits

The general scaled addition (using MUX) in SC for N_{in} inputs follow (13) which is easily realized using $N_{in} - 1$ 2-to-1 MUXs tree with uncorrelated selector bit-streams of probability 0.5 for each of the MUXs. Thus, average Pooling is straightforward in SC. For example, if we want to implement in SC average pooling with stride size 2 and kernel size 2, 3 scaled addition units

(2-to-1 MUXs) tree with selector probability $\mathrm{p} = 0.5$ will be used. To increase the accuracy, TFF will be utilized for selector bit-stream generation. The average pooling block can be used with any SCC among inputs. Two versions of average pooling will be experimented, the average pooling using independent RNGs for MUX selectors (SC AP RNG) and the average pooling using T-FF to create the uncorrelated selector bit-stream (SC AP FF).

$$z = \frac{1}{N_{in}} \sum_{i=0}^{N_{in}-1} x_i \tag{13}$$

5. Experimental Results and Discussion

To clarify the effectiveness of the proposed SC CNN basic functions, they were compared with previous SC work and the respective binary computation. The accuracy and the resource utilization are the measured metrics. To evaluate the accuracy, the absolute error is computed for 10000 attempts of randomly generated inputs where the conventional binary result is the golden reference. From these attempts, we obtained the average output absolute errors. Then different SN lengths L were used to observe the error behavior and the robustness to rounding error as the input binary precision $\mathrm{n} = 32$ bits. On the other hand, to evaluate the area of the basic functions designs, we synthesize the circuits using Vivado Design Suite targeting Xilinx ZYNQ Z706 FPGA.

Previous SC CNN used XNOR-MUX or XNOR-APC for the inner product operation in convolutional layers of CNN [7, 8, 10]. This work proposed MUX tree for the inner product shown in Figure 5, and the selector probability values follow (8) [21]. The number of inputs N_{in} used in this experiment is 16 since it is more optimum for XNOR-APC SC inner product. The SN length is varied through 64, 128, 256, 512, 1024, 2048, 4096, and 8192 bits since 8192 bits is used in [10] and 1024 bits in [8]. Figure 6(a) shows the mean absolute error of MUX tree, XNOR-MUX, and XNOR-APC approaches for inner product. To make a fair comparison, the results of each SC circuit is multiplied by its scaling factor. For example, the MUX tree output is multiplied by $\sum|h|$. The MUX tree obtained the least error. Therefore, the MUX tree SC inner product is more accurate than previous works approaches with respect to different SN lengths.

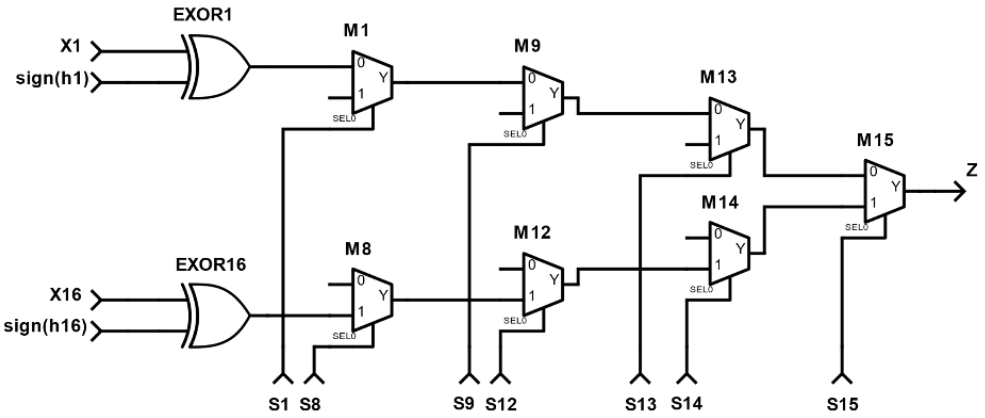


Figure 5. MUX tree used for 4×4 inner product

The resource utilizations of the three SC inner product approaches MUX tree, XNOR-MUX, and XNOR-APC are compared along with conventional binary (bin) inner product (serial design) as shown in Table 3. The MUX tree shows lower resource utilization of $1.6\times$ and $2\times$ compared to XNOR-MUX and XNOR-APC. Also, significant savings compared to the binary inner product. Besides, MUX tree has more area savings since the MUX tree circuit requires one RNG for any number of inputs. However, XNOR-MUX and XNOR-APC SC inner product need N_{in} RNGs. Therefore, the MUX tree obtains $\times N_{in}$ RNG circuit savings. The RNG used is linear feedback shift register LFSR.

Table 3. Resource Utilization of Proposed Functions, Previous Work, and Conventional Binary

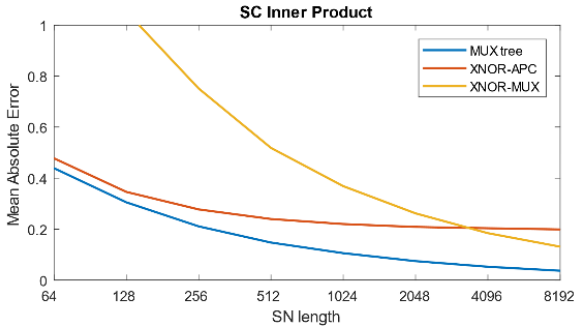
Inner Product	DSP	FF	LUT
MUX tree	0	0	35
XNOR-MUX	0	0	55
XNOR-APC	0	0	71
Bin 8-bit fixed point	1	26	18
Bin 32-bit float	5	540	740

Activation function	DSP	FF	LUT
Proposed SC ReLU	0	0	1
Bin ReLU 32-bit	0	0	16

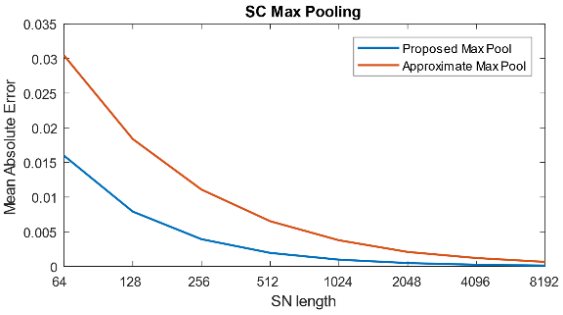
Max Pooling (MP)	DSP	FF	LUT
Proposed SC MP	0	0	1
Approximate SC MP [10]	0	32	65
Bin MP 32-bit	0	0	134

Average Pooling (AP)	DSP	FF	LUT
Proposed SC AP FF	0	3	4
SC AP RNGs	0	0	2
Bin AP 32-bit	0	0	95

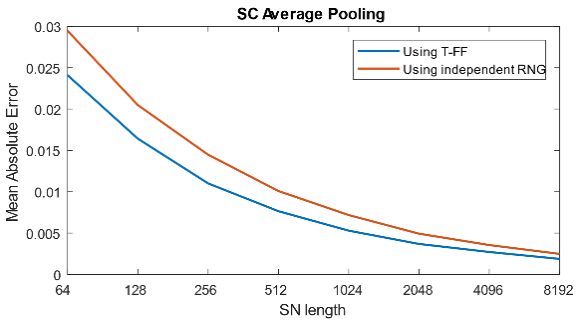
Using the proposed MUX tree for SC inner product operation of the CNN convolutional layer is more efficient compared to previous SC inner product circuits or the conventional binary. MUX tree is more accurate than other SC inner product and has less hardware footprint. Also, compared to conventional binary, using the MUX tree SC inner product will provide significant resource utilization savings. Without exploiting correlation, the max pooling operation in SC is hard to be designed. Ren et al. [10] proposed an approximate SC max pooling circuit. However, the proposed SC max pooling in this study outperforms the previous work in terms of accuracy and resource utilization. Figure 6 (b) shows that the proposed max pooling is more accurate for any SN length L. Also Figure 6 (c) shows the absolute error of proposed average pooling operation using independent RNG for each MUX selector and T-FF for selector bitstream generation. The average pooling using independent RNGs requires $log_2(N_{in}) + 1$ different RNGs, while the average pooling using T-FF and MUXs require 1 RNG for any number of inputs. This result a $(log_2(N_{in}) + 1)$ times savings in RNGs. The resource utilization of the proposed SC average and max-pooling functions and their binary counterparts are shown in Table 3 where all are of parallel architecture. The proposed SC ReLU circuit absolute error is shown in Figure 6(d). A very minimal accuracy loss in the proposed SC ReLU is obtained with a high resource utilization savings of 16 times.



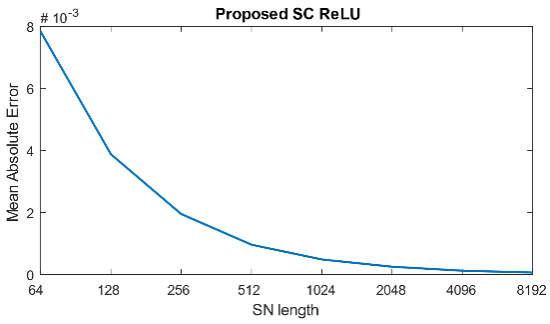
(a) SC inner product circuits absolute error



(b) SC max pooling absolute error



(c) SC average pooling absolute error



(d) SC ReLU absolute error

Figure 6. The absolute error of the proposed basic functions compared to previous works

6. Conclusion

In this paper, the SC CNN basic functions exploiting correlation was proposed with reduced hardware footprint to be efficient in the resource-constrained mobile and embedded systems. These functions are inner product, max pooling, average pooling, and ReLU activation function. A combination of these basic functions when looped create a specific CNN layer. Experimental results demonstrate that the proposed SC functions achieved significant hardware footprint savings compared to equivalent binary functions. Also, the proposed functions outperformed previous works of SC CNN in terms of accuracy and resource utilization. Our future work will investigate the performance of a complete SC CNN which is composed of the proposed basic functions.

References

[1] Y LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*. 2015; 521(7553)436–444.
[2] B Cheung. *Convolutional Neural Networks Applied to Human Face Classification.* 2012 11th Int. Conf. Mach. Learn. Appl. 2012: 580–583.
[3] C Wu, W Fan, Y He, J Sun, S Naoi. *Cascaded Heterogeneous Convolutional Neural Networks for Handwritten Digit Recognition*. Int. Conf. Pattern Recognit*.* 2012: 657–660.
[4] J S Ashwin, N Manoharan. Convolutional Neural Network Based Target Recognition for Marine Search. *Indones. J. Electr. Eng. Comput. Sci.* 2017; 8(2): 561–563.
[5] R Wang, J Zhang, W Dong, J Yu, C J. Xie, R Li, T Chen, H Chen. A Crop Pests Image Classification Algorithm Based on Deep Convolutional Neural Network. *Telkomnika Telecommunication Computing Electronics and Control.* 2017; 15(3): 1239–1246.
[6] M Alawad, M. Lin. Stochastic-Based Deep Convolutional Networks with Reconfigurable Logic Fabric. *IEEE Trans. Multi-Scale Comput. Syst.* 2016; 99: 1.
[7] J Li, A Ren, Z Li, C Ding, B Yuan, Q Qiu, Y Wang. *Towards acceleration of deep convolutional neural networks using stochastic computing.* Proc. Asia South Pacific Des. Autom. Conf. ASP-DAC. 2017: 115–120.
[8] H Sim, D Nguyen, J Lee, K. Choi. *Scalable stochastic-computing accelerator for convolutional neural networks.* Proc. Asia South Pacific Des. Autom. Conf. ASP-DAC. 2017: 696–701.
[9] A Alaghi, J P Hayes. *Exploiting correlation in stochastic circuit design*. 2013 IEEE 31st Int. Conf. Comput. Des. ICCD. 2013: 39–46.
[10] 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. 2017: 405–418.
[11] 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*. in International Joint Conference on Neural Networks (IJCNN). 2017: 1230–1236.
[12] W Qian, X Li, M D Riedel, K Bazargan, D J Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Trans. Comput.* 2011*;* 60(1): 93–105.
[13] A Krizhevsky, I Sutskever, G E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.* 2012*:* 1–9.
[14] S-M Khaligh-Razavi. *What you need to know about the state-of-the-art computational models of object-vision: A tour through the models*. arXiv Prepr. arXiv1407.2776. 2014: 36.
[15] C Zhang, P Li, G Sun, Y Guan, B Xiao, . Cong. *Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks.* Proc. 2015 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays - FPGA*.* 2015; 15: 161–170.
[16] C Szegedy, W Liu, Y Jia, P Sermanet, S Reed, D Anguelov, D Erhan, V Vanhoucke, A Rabinovich. *Going Deeper with Convolutions*. in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015: 1–9.
[17] A Alaghi and J P Hayes. Survey of Stochastic Computing. *ACM Trans. Embed. Comput. Syst.* 2013; 12(2s): 1–19.
[18] B. Gaines. Stochastic computing systems. *Adv. Inf. Syst. Sci.* 1969: 37–172.
[19] V C Gaudet, A C Rapley. Iterative decoding using stochastic computation. *Electron. Lett.* 2003; 39(3): 299–301.
[20] K Kim, J Lee, K Choi. *Approximate de-randomizer for stochastic circuits*. *ISOCC 2015 - I*nt. SoC Des. Conf. SoC Internet Everything. 2016: 123–124.
[21] H Ichihara, T Sugino, S Ishii, T Iwagaki, T Inoue. Compact and Accurate Digital Filters Based on Stochastic Computing. *Trans. Emerg. Top. Comput.* 2016; 99: 1–12