DSCNN: Hardware-Oriented Optimization for Stochastic Computing Based Deep Convolutional Neural Networks

Zhe Li*, Ao Ren*, Ji Li†, Qinru Qiu*, Yanzhi Wang*, Bo Yuan‡
*Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13224, USA
{zli89, aren, qiqiu, ywang393}@syr.edu
†Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089, USA
jli724@usc.edu
‡ Department of Electrical Engineering, City University of New York, City College, New York, NY 10031, USA
byuan@ccny.cuny.edu

***Abstract*—Deep Convolutional Neural Networks (DCNN), a branch of Deep Neural Networks which use the deep graph with multiple processing layers, enables the convolutional model to finely abstract the high-level features behind an image. Large-scale applications using DCNN mainly operate in high-performance server clusters , GPUs or FPGA clusters; it is restricted to extend the applications onto mobile/wearable devices and Internet-of-Things (IoT) entities due to high power/energy consumption. Stochastic Computing is a promising method to overcome this shortcoming used in specific hardware-based systems. Many complex arithmetic operations can be implemented with very simple hardware logic in the SC framework, which alleviates the extensive computation complexity. The exploration of network-wise optimization and the revision of network structure with respect to stochastic computing based hardware design have not been discussed in previous work. In this paper, we investigate *Deep Stochastic Convolutional Neural Network (DSCNN)* for DCNN using stochastic computing. The essential calculation components using SC are designed and evaluated. We propose a joint optimization method to collaborate components guaranteeing a high calculation accuracy in each stage of the network. The structure of original DSCNN is revised to accommodate SC hardware design's simplicity. Experimental Results show that as opposed to software inspired feature extraction block in DSCNN, an optimized hardware oriented feature extraction block achieves as higher as 59.27% calculation precision. And the optimized DSCNN can achieve only 3.48% network test error rate compared to 27.83% for baseline DSCNN using software inspired feature extraction block.**

***Keywords*—Deep Learning; Deep Convolutional Neural Networks; Stochastic Computing; Hardware-oriented Co-optimization**

I. Introduction

Convolutional Neural Networks (CNNs) have been proved as an effective series of models for analyzing, abstracting, and understanding image contents. In recent years, *deep convolutional neural networks (DCNN)*, a branch of *Deep Neural Network (DNN)* which use the deep graph with multiple processing layers [1], enable the convolutional model to finely abstract the high-level features behind an image. The applications range from document recognition [2], [3] to face detection [4], from image/video classification [5–7], [8] to object detection [9], [10]. DCNN is now the dominant approach for almost all recognition and detection tasks, and approaches human performance on some tasks [1]. Nevertheless, deep graph of layers in DNN has brought about significant increases in computation complexity. Large-scale applications using DCNN mainly operate in high-performance server clusters , GPUs or FPGA clusters [11–14]; it is restricted to extend the applications onto mobile/wearable devices and *Internet-of-Things (IoT)* entities due to high power/energy consumption.

Aforementioned efforts focused on improving computing capability for DCNN and highly relied on the development of hardware devices. The rate of progress, however, will saturate gradually in next decade due to Moore's law's fading foreseen by Gordon Moore himself [15]. An urgent need of new computation technology encourages researchers' further investigation. *Stochastic Computing (SC)* is a promising method to overcome this shortcoming used in specific hardware-based systems, which has ultra-low footprint and inherent fault tolerance against soft errors [16], [17]. As a unique data representation and processing technique, SC enables the design of fully-parallel and scalable hardware implementations of large-scale deep networks. Many complex arithmetic operations can be implemented with very simple hardware logic in SC the framework [18], which alleviates the extensive computation complexity. This offers an immense design space for (i) neuron integrations due to the significantly reduced area per neuron and (ii) performance optimizations with respect to resiliency by trading off power/energy, speed, and area budget.

Previous works in [19–21] have validated SC's capability in neural network computation. The exploration of network-wise optimization and the revision of network structure with respect to stochastic computing based hardware design were not discussed in aforementioned researches. In this paper, we investigate LeNet-5 [2] as a DCNN case study. We call the DCNN using stochastic computing as *Deep Stochastic Convolutional Neural Network (DSCNN)*. The essential calculation components using SC are designed and evaluated. We propose a joint optimization method to collaborate components guaranteeing a high calculation accuracy in each stage of the network. The structure of original DSCNN is revised to accommodate SC hardware design's simplicity.

The contribution of this paper is summarized as follows,

- We analyzed hardware components' properties and designed the essential components used in DSCNN inference calculation.
- We concluded optimization functions to jointly configure components' parameters in DSCNN to guarantee a high calculation precision.
- The software inspired structure of DSCNN is revised and evaluated. Incorporating with the proposed optimization function, the hardware-oriented DSCNN achieves a high calculation precision and low network test error rate.

The rest of this paper is organized as follows. Section II presents the modularization of DCNN and the SC implementation of each module in DSCNN. Section III shows the proposed joint optimization of DSCNN modules and structural optimization of DSCNN. Finally, Section IV summarizes the work.

Zhe Li* and Ao Ren* contributed equally to this work

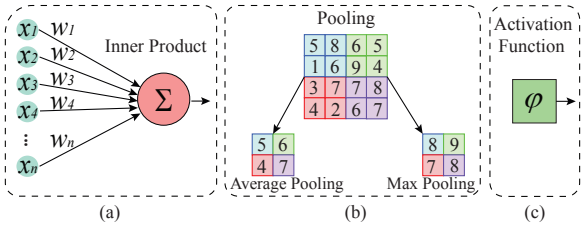

Fig. 1. Neurons in a DCNN. (a) Inner Product, (b) pooling, and (c) activation

II. Modularization of DCNN and its SC Implementation

A. Deep Convolutional Neural Network Architecture

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

After obtaining features using convolution, a subsampling step can be applied to aggregate statistics of these features to reduce the dimensions of data and mitigate over-fitting issues. This subsampling operation is realized by a *pooling neuron* in pooling layers, where different non-linear functions can be applied, such as max pooling, average pooling, and L2-norm pooling.

The activation functions are non-linear transformation functions, such as Rectified Linear Units (ReLU) $f(x) = max(0, x)$, hyperbolic tangent (tanh) $f(x) = tanh(x)$ or $f(x) = |tanh(x)|$, and sigmoid function $f(x) = \frac{1}{1+e^{-x}}$. Usually, a combination of convolutional neurons, pooling neurons, and activation functions forms a *feature extraction block* to extract high-level abstraction from the input images or previous low level features.

The fully connected layer is a normal neural network layer with its inputs fully connected with its previous layer. Each *fully connected neuron* calculates the inner product of its inputs and corresponding weights. The loss function of a DCNN specifies how the network training penalizes the deviation between the predicted and true labels, and typical loss functions are softmax loss, sigmoid cross-entropy loss or Euclidean loss.

In general, we can conclude three kinds of basic calculations in a DCNN based on their corresponding operations as Fig.1 shows. Neurons in convolutional layers and fully connection layers calculate the inner product shown in Fig.1 (a) of inputs and weights based on its incoming connection with the previous layer. And the products are subsampled through a pooling neuron shown in Fig.1 (b). The subsampled outputs are transformed by an activation function shown in Fig.1 (c) to ensure the inputs of next layer are within the valid range.

B. Stochastic Computing based components

Stochastic computing is a technology that represents a probabilistic number in the range of $[0, 1]$ by counting the number of ones in a bit-stream. For instance, the bit-stream 0100110100 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 the bipolar encoding scheme, a real number x is processed by $x = 2P(X = 1) - 1$ i.e. $P(X = 1) = \frac{x+1}{2}$, thus 0.4 can be represented by 1011011101.

Previous works [19], [20] haven briefly introduced three basic arithmetic operations in neural network inference process, including addition, multiplication, and hyperbolic tangent (tanh). These stochastic based operations are implemented with much lower hardware cost compared to conventional binary computing. As mentioned in Section II-A, calculations in a DCNN inference process includes inner product, pooling, and activation function. Hardware designs of these three calculations are introduced as follows in this section. Please note that we use bipolar encoding format for the design, for numbers in a DCNN are distributed on both sides of zero.

1) MUX-based Inner Product Calculation Unit: The inner production calculation is composed of multiple multiplications and one addition. Multiplications are conducted by XNOR gates [19] to generate products of each pair of inputs. We use an n-to-1 MUX to sum all the aforementioned products up; the result of the MUX is the inner product of a pair of vectors with a scaling down factor of $\frac{1}{n}$. This is because every bit of the output is randomly selected from n input bits; the probability of each input to be selected is $\frac{1}{n}$. This the inherent down-scaling property of MUX. From the perspective of stochastic computing, the output bit-stream represents a number of $\frac{1}{n}(x_0w_0 + x_1w_1 + \cdots + x_{n-1}w_{n-1})$.

2) Average pooling: Pooling, also known as down-sampling or sub-sampling, significantly reduce the number of neurons in next layer in a DCNN. The inter-layer connections are reduced and the invariance of the extracted features are maintained. As Fig. 1(b) shows, in most DCNN structures, each 2×2 region of pixels in feature maps is down-sampled to be one pixel. Average pooling calculates a mean of a small matrix, thus similarly the inherent down-scaling property of the MUX mentioned in Section II-B1 is used to average stochastic numbers. Thus 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.

3) Finite State Machine (FSM) based Hyperbolic Tangent (tanh): Authors in [19] proposed a K-state FSM-based design *Stanh* for tanh. A relationship between Stanh and tanh was given as $Stanh(K, x) \cong tanh(\frac{K}{2}x)$ with input x. An input distributed in $[-\frac{K}{2}, \frac{K}{2}]$ in tanh is mapped to $[-1, 1]$ in Stanh guaranteeing this mapped number to be represented by a bipolar stochastic bit-stream. Claimed in [19], the FSM design achieved better accuracy with increased state number K,.

TABLE I
The Relationship Between State Number and Relative Error of Stanh with bit-stream length = 8192

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

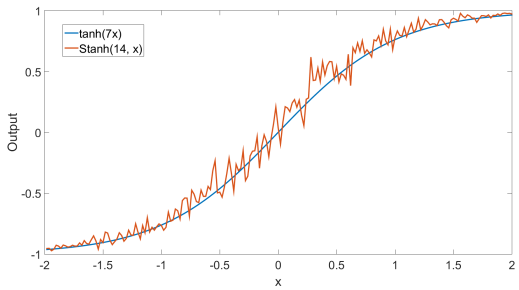

Fig. 2. Stanh v.s. tanh results with state $K = 14$ and bit-stream length $L = 8192$

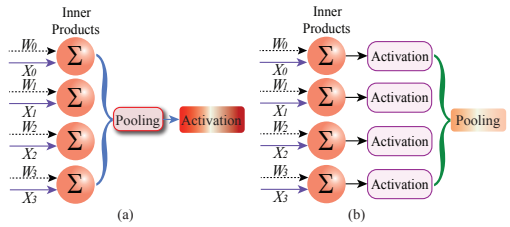

Fig. 3. (a) software-based feature extraction block, (b) hardware-oriented feature extraction block

However, this conclusion cannot br applied to our work for three reasons: (i) as shown in Table. I, when the input of tanh is distributed in $[-1, 1]$ instead of $[-\frac{K}{2}, \frac{K}{2}]$, the accuracy is not linearly proportional to K; (ii) the aforementioned conclusion resulted from the assumption that x is precisely represented, which means the bit-stream must be very long. But when the bit-stream is not impractical long, the bit-stream length must be taken into consideration to refine the equation; (iii) one can observe in Fig. 2, when the input is closer to zero, the error between tanh and Stanh gets bigger. So a scaled-down input to Stanh must be scaled back to $[-1, 1]$ to reduce error. Hence, equation $Stanh(K, x) \cong tanh(\frac{K}{2}x)$ must be optimized considering bit-stream length and scaling factor of the inputs. We proposed a optimized function $K = f(L, N)$ where L is the length of bit-stream and N is the fan-in, and the optimized parameters will be discussed in Section III.

III. Hardware Oriented Optimization for DSCNN

In software based algorithm design, a *feature extraction block* of DCNN is formed by convolution neurons, pooling neurons and activation funtion in order as Fig. 3(a) shows. This is reasonable, because intuitively the order of pooling before activation can save 3/4 computation resources to do the activation. However, in hardware design, due to the cross-dependency of components, another arrangement of neurons (pooling after activation shown in Fig. 3(b)) in a feature extraction block must be investigated. In this section, we investigate two different arrangements as shown in Fig. 3(a) and Fig. 3(b) using MUX-based inner product calculation unit (in short, *MUXIP*), Average pooling (*AVG*), and FSM-based Stanh (*STANH*).

A. MUXIP-AVG-STANH

As mentioned in Section II-B3, when Stanh is utilized, the state number needs to be carefully selected with a comprehensive consideration of the scaling factor and bit-stream length. Below is the empirical equation that is extracted from immense experiments to obtain the approximately optimal state number providing a high accuracy.

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

where $N = 1/s$ which is the input size of the feature extraction block (as well as the size of weights), L is the bit-stream length. γ is the empirical parameter which is 33 in our case. Given N and L, the state number K is the nearest even number of the value of the right side of above equation.

We evaluated the average absolute error of non-optimized and optimized feature extraction blocks given $10,000$ sets of random inputs ranging from -1 to 1 with different input sizes and bit-stream lengths. The average absolute error is the mean of absolute differences between the expected results and the observed results for the same test cases. Shown in Fig. 4, we observed for the non-optimized feature extraction block:

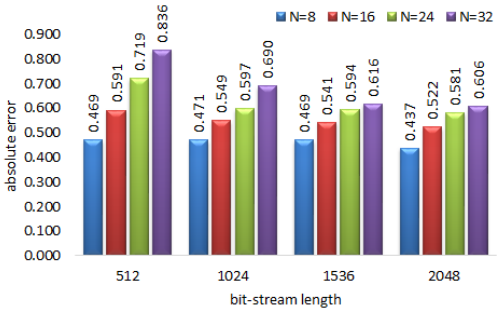

Fig. 4. Absolute error for non-optimized feature extraction block

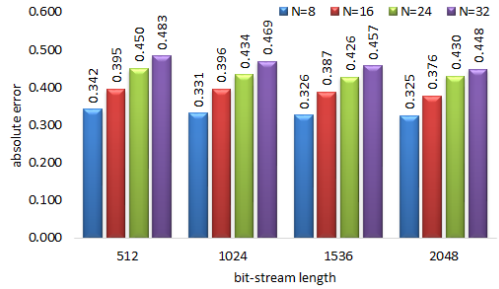

Fig. 5. Absolute error for optimized feature extraction block

(i) As bit-stream gets longer, the absolute error decreases for the same input size N. Because a number can be represented more precisely with longer bit-stream. But the improvements are not significant. (ii) With the same length of bit-stream, more inputs lead to larger absolute error.

Similarly, we evaluated the average absolute errors with different input sizes and bit-stream lengths for an optimized feature extraction block applying eqn.1 to set up the FSM state number in Stanh. As Fig. 5 shows, compared with the error by the same configuration of bit-stream length and input size respectively in Fig.4, the optimized feature extraction blocks achieves as high as 42.22% better accuracy.

B. MUXIP-STANH-AVG

Conventional feature extraction block design down-samples the inner products first and then calculates the activation function for the down-sampled value, which is described in Section III-A. However, this design is not the optimal for DSCNN in reality. We take handwritten digits recognition using LeNet-5 [2] as example. LeNet-5 consists of two consecutive feature extraction layers followed by a fully-connected layer. We investigate the influences of errors in the first (Layer-0) and second (Layer-1) feature extraction layers on the overall test error rate of the entire DCNN. Then the combined impact of Layer-0 and Layer-1 on the overall test error rate is also explored. Shown in Fig. 6,the absolute errors of feature extraction blocks in each layer follow a normal distribution. Given various standard deviation of error distribution, we evaluate the test error rate for complete DSCNN. The handwritten digit image dataset consisting of $60,000$ training data and $10,000$ testing data with 28x28 grayscale image and 10 classes is used in the experiments. We trained the network model to achieve a network test error of 1.54% as the reference.

It is observed in Fig. 6 that a layer closer to the inputs has a more serious impact on the overall accuracy of the DCNN than a layer closer to the output layer. Because inaccurate features captured near the inputs may affect all the following layers, whereas the errors occurring near the output layer can only disturb a few subsequent layers. It also shows that with the error range of each feature extraction block increase,

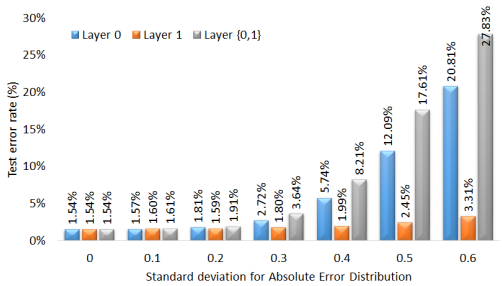

Fig. 6. Test error rate of LeNet-5 with inaccurate Layer-0,Layer-1,and Layer-0,1 together

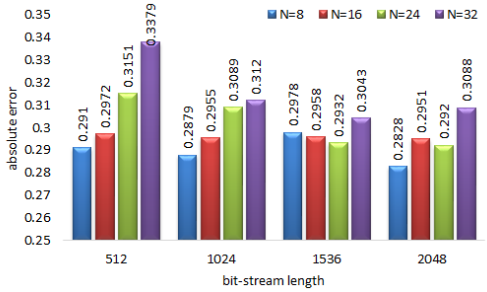

Fig. 7. Absolute error for optimized revised feature extraction block

test error rates with inaccurate Layer-0 and 1 together gives unacceptable results ($\geq 5\%$ error rate). When the error of each layer ranges from $[-0.3, 0.3]$, errors in Layer-0 itself gives the network error rate of 5.74%. and the errors in both feature extraction layers lead to an 8.21% error rate. Considering each feature extraction block in LeNet-5 takes 25 inputs, intuitively the error in each layer is about 0.45 estimated from Fig. 5 which leads to network test error rate of $[8.21\%, 17.61\%]$ from Fig. 6. What is worse, for the non-optimized feature extraction block design, each of which has an absolute error of about 0.6, the network test error rate is 27.83% based on Fig. 6.

To improve the optimization method proposed in Section III-A , we revised the conventional design by switching the order of AVG and STANH, which is shown in Fig. 3(b). Similarly, we also propose an empirical equation to get an optimized state number K of stanh unit as follows,

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

where $\alpha = 1.3$ and $\beta = 8.74$ which are empirical parameters. $N = 1/s$ is the input size of the feature extraction block (as well as the size of weights), L is the bit-stream length. Given N and L, the state number K is the nearest even number of the value of the right side of above equation.

As shown in Fig. 7, we have a similar but smaller error distribution compared to Fig. 5. The revised feature extraction block structure achieves as high as 33.48% better accuracy compared to conventional optimized design and 59.57% better compared to conventional non-optimized design. We evaluated impact of the optimized revised feature extraction block on entire DCNN. Given $N = 25$(each filter is 5×5 pixels) and $L = 512, 1024, 1536, 2048$, the network test error rates are 3.97%, 3.65%, 3.53%, 3.48%.

IV. Conclusion

This paper introduced hardware oriented structural optimization for Deep Convolutional Neural Network using Stochastic Computing. The first optimization method relied on the characteristics of Stochastic Computing based component; it concluded a joint optimization equation, by which we achieved as higher accuracy as 42.22% compared to the software inspired feature extraction block in the DSCNN without optimized parameters of the components. The second optimization method broke out the stereotype in software-inspired DSCNN; it restructured the order of neurons in a feature extraction block in DSCNN to reduce the absolute error by 33.48% compared with the first optimization and 59.57% compared to non-optimized baseline. Experimental results showed that the optimized hardware-oriented feature extraction block can achieve the network error rate as low as 3.48% while the non-optimized baseline only achieved the test error of 27.83%.

References

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

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

[3] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[4] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

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

[6] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.

[7] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, p. 1237.

[8] 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*, 2014, pp. 1725–1732.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[11] B. Catanzaro, "Deep learning with cots hpc systems," 2013.

[12] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[13] 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*. ACM, 2014, pp. 675–678.

[14] 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*. Citeseer, 2011.

[15] R. Courtland, "Gordon moore: The man whose name means progress," *IEEE Spectrum*, vol. 30, 2015.

[16] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.

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

[18] B. R. Gaines, "Stochastic computing systems," in *Advances in information systems science*. Springer, 1969, pp. 37–172.

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

[20] 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.

[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*. ACM, 2016, p. 124.