Structural Design Optimization for Deep Convolutional Neural Networks using Stochastic Computing

Zhe Li*, Ao Ren*, Ji Li†, Qinru Qiu*, Bo Yuan‡, Jeffrey Draper§†, Yanzhi Wang*
*Syracuse University, Syracuse, USA ({zli89, aren, qiqiu, ywang393}@syr.edu)
†University of Southern California, Los Angeles, USA (jli724@usc.edu)
‡City University of New York, New York, USA (byuan@ccny.cuny.edu)
§Information Sciences Institute, Marina Del Rey, USA (draper@isi.edu)

***Abstract*—Deep Convolutional Neural Networks (DCNNs) have been demonstrated as effective models for understanding image content. The computation behind DCNNs highly relies on the capability of hardware resources due to the deep structure. DCNNs have been implemented on different large-scale computing platforms. However, there is a trend that DCNNs have been embedded into light-weight local systems, which requires low power/energy consumptions and small hardware footprints. Stochastic Computing (SC) radically simplifies the hardware implementation of arithmetic units and has the potential to satisfy the small low-power needs of DCNNs. Local connectivities and down-sampling operations have made DCNNs more complex to be implemented using SC. In this paper, eight feature extraction designs for DCNNs using SC in two groups are explored and optimized in detail from the perspective of calculation precision, where we permute two SC implementations for inner-product calculation, two down-sampling schemes, and two structures of DCNN neurons. We evaluate the network in aspects of network accuracy and hardware performance for each DCNN using one feature extraction design out of eight. Through exploration and optimization, the accuracies of SC-based DCNNs are guaranteed compared with software implementations on CPU/GPU/binary-based ASIC synthesis, while area, power, and energy are significantly reduced by up to 776×, 190×, and 32835×.**

I. Introduction

Deep Convolutional Neural Networks (DCNNs) have been demonstrated as effective models for understanding image content [1], due to their special structural designs [2] of layer-wise local connections implementing convolution, integrating pattern matching techniques into neural networks, and learning invariant elementary features of images. Researchers has made significant achievements utilizing DCNNs in image classification [3], video classification [1], and object detection [4].

The computation behind DCNNs highly relies on the capability of hardware resources due to the deep structure. From high performance server clusters [5] to *General-Purpose Graphics Processing Units (GPGPUs)* [6], parallel accelerations of DCNNs are widely used in both the academic and industrial world. Moreover, hardware oriented acceleration for DCNNs has attracted enormous research interests on *Field-Programmable Gate Arrays (FPGAs)* [7]. Nevertheless, there is a trend that DCNNs have been embedded into light-weight local systems, which requires low power/energy consumptions and small hardware footprints [8], such as self-driving systems [9], and wearable devices [10]. A promising technique that guarantees smaller hardware footprints with limited power/energy budget is urgently needed [11].

Stochastic Computing (SC), as a low-cost substitute to binary-based computing [12], radically simplifies the hardware implementation of arithmetic units and has the potential to satisfy the small low-power needs of DCNNs. Many complex arithmetic operations can be implemented with very simple hardware logic in a stochastic computing framework, alleviating the extensive computation complexity, and offering a colossal design space for integration and optimization due to its reduced area and error resiliency [13]. Predecessors in [12,14,15] investigated SC's applications on neural networks and *Deep Belief Networks (DBNs)*, which validated the possibility to design DCNNs using stochastic components.

However, unlike DBNs, local connectivities and down-sampling operations [2] have made DCNNs more complex to be implemented using SC. Moreover, software experience is not in accordance with hardware implementations for SC-based DCNNs. The contributions of this paper are concluded as follows, *1)* We investigated previous hardware designs using SC for DCNN and optimized essential components which have never been studied in existing researches; *2)* We explored eight structural designs for blocks in DCNN using SC. We proposed optimization jointly configuring hardware components' parameters guaranteeing a high calculation precision for each design; *3)* We incorporated each aforementioned block design into a complete DCNN. Network accuracy performance was evaluated showing a close performance to a software-based DCNN. Hardware synthesis results demonstrated that proposed hardware designs for DCNNs were able to achieve a smaller hardware footprint and lower power/energy.

II. Overview of Deep Convolutional Neural Network architecture

In spite of an input layer and an output layer, a general DCNN architecture consists of a stack of *convolutional layers*, *pooling layers*, and *fully connected layers*. In general, we can conclude three kinds of basic *function blocks* in a DCNN inference process 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 corresponding to their incoming connections with the previous layer. Please note convolution is essentially a calculation of inner-product. And the products are sub-sampled through a pooling neuron shown in Fig.1(b) either by an average pooling or a max pooling. The inner-products or sub-sampled outputs are transformed by an activation function shown in Fig.1(c) to ensure the inputs of next layer are within the valid range. Typical activation functions are Rectified Linear Units (ReLU), hyperbolic tangent (tanh), and sigmoid.

Usually, a combination of convolutional neurons, pooling neurons, and activation functions forms a *feature extraction block (FEB)* to extract high-level abstraction from the input images or previous low-level features. The overall network accuracy (e.g., the overall classification rates) is one of the

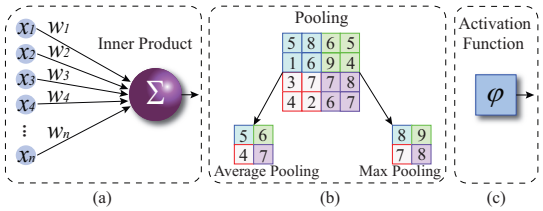

Fig. 1. Neurons in a DCNN. (a) Inner-Product, (b) pooling, and (c) activation

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

key optimization goals of the SC-based DCNN. On the other hand, the SC-based function blocks and FEBs exhibit a certain degree of imprecision due to the inherent stochastic nature. The network accuracy and hardware precision are different but correlated, i.e., the high precision in each function block will likely lead to a high overall network accuracy. Hence, the hardware precision should be optimized in the design of SC-based function blocks and FEBs.

III. Component Design using Stochastic Computing

In this paper, we adopt the bipolar encoding format of stochastic computing, in which numbers in the range of $[-1, 1]$ are represented by bit-streams. To be more specific, a real number x is processed by $x = 2P_{X=1} - 1$, where $P_{X=1}$ indicates the probability of ones in bit-stream X. For instance, 0.4 can be represented by 1011011101.

A. Inner-Product Calculation Blocks

The inner-product calculation is composed of multiple multiplications and one addition as it is shown in Fig 1(a). Multiplications are conducted by XNOR gates [12] to generate products of pairs of input and weight using bipolar encoding format. In SC domain, the addition is accomplished by various possible hardware implementations including *Approximate Parallel Counter (APC)* [14] based and *Multiplexer (MUX)* based adders. Details about APC-based inner-product calculation block can be found in [14]. As for MUX-based inner-product calculation block, an n-to-1 MUX is used to sum up all the products; the result of the MUX is the inner-product of a pair of vectors with a scaling down factor of $1/n$, since every bit of the output is randomly selected from n input bits; the probability of each input to be selected is $1/n$. This is the inherent down-scaling property of MUX. From the perspective of stochastic computing, the output bit-stream represents a number of $\frac{1}{n}\sum_0^{n-1} x_i w_i$ in Fig. 1(a) as an example.

B. Pooling Blocks

Average pooling was often used in earlier researches of CNN [11] while max pooling has risen in favor due to its better performance in practice. We investigate hardware implementations for both pooling schemes. As Fig. 1(b) shows, for instance, 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 III-A is used to average stochastic numbers, i.e., a 4-to-1 MUX is used to calculate the mean of four bit-streams. Max pooling, on the other hand, tries to find the maximum value in a small region. However, in the stochastic domain, the determination of magnitude must be made after counting the whole bit streams, which results in very long latency and a waste of energy. In this paper, we adopt the hardware-oriented max pooling design developed in [8], where the largest bit-stream in the most recent segment is selected as the near-max output.

C. Activation Function Blocks

We investigate two tanh activation designs in this paper. The original design of Finite State Machine (FSM) based stochastic hyperbolic tangent (Stanh) block was proposed by the authors in [12]. 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. On the other hand, binary number based hyperbolic tangent (Btanh) block is proposed in [14], which is implemented by a saturated up/down counter to convert the binary outputs of the APC-based inner-product calculation block to bit-stream in SC domain.

TABLE I
The designs of FEBs and corresponding optimization functions

No.	Design	Optimization Function	Parameters
1	MUXIP-AVG-STANH	$K = f(L,N) \approx 2\log_2(N) + N\log_2(L)/\gamma\log_2(N)$	$\gamma = 33.27$
2	MUXIP-STANH-AVG	$K = f(L,N) \approx \alpha\log_2 N + N\log_5 L/\beta\log_2 N$	$\alpha = 1.3$ $\beta = 8.74$
3	MUXIP-NMAX-STANH	$K = f(L,N) \approx 2(\log_2 N + \log_2 L) - \alpha/\log_2 N - \beta/\log_5 L$	$\alpha = 37$ $\beta = 16.5$
4	MUXIP-STANH-NMAX	$K = f(L,N) \approx -\gamma\sqrt{N}\log_2 N/L + \alpha\log_2 N + L/\beta\log_2 L$	$\alpha = 1, \beta = 5$ $\gamma = 5.2$
5	APCIP-AVG-BTANH	$K = g(N) \approx \alpha N$	$\alpha = 0.5$
6	APCIP-BTANH-AVG		$\alpha = 2$
7	APCIP-NMAX-BTANH		
8	APCIP-BTANH-NMAX		

IV. Structural Designs for DCNN using SC

A. Co-Optimization of Stanh

Claimed in [12], the FSM design achieved better precision with increased state number K. However, based on the evaluation in [16], this conclusion cannot be applied to our work for three reasons: (i) when the input of tanh is distributed in $[-1, 1]$ instead of $[-\frac{K}{2}, \frac{K}{2}]$, the precision 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) MUX-based inner product blocks will scale down the real inner product values, so a scaled-down input to Stanh must be scaled back to $[-1, 1]$ by selecting a proper state number K. 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 an 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 later in this section.

B. Proposed Structural Designs of Feature Extraction Blocks

In software-level design, a FEB of DCNN is formed by convolution neurons, pooling neurons and activation function in order.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 and their effects on calculation precision, another arrangement of neurons (pooling after activation) in an FEB must be investigated [17]. In this section, we investigate two different arrangements.

Eight designs of FEBs are analyzed and optimized listed in Table I by permuting MUX-based inner-product calculation block (in short, *MUXIP*), APC-based inner-product calculation block (*APCIP*) Average pooling (*AVG*), Near-Max Pooling (NMAX), FSM-based Stanh (*STANH*) and Binary-based Btanh (*BTANH*). For each design, we extract empirical functions by regression of exhaustive data samples, which is shown in Table I. The enormous data samples are generated randomly and the expected outputs are regarded as golden references for the regression functions respectively. The functions are obtained by minimizing the difference between golden reference and the calculated value of a design with a specific K.

As shown in Table I, given the input size N (and bit-stream length L for MUXIP based designs), an optimal number of

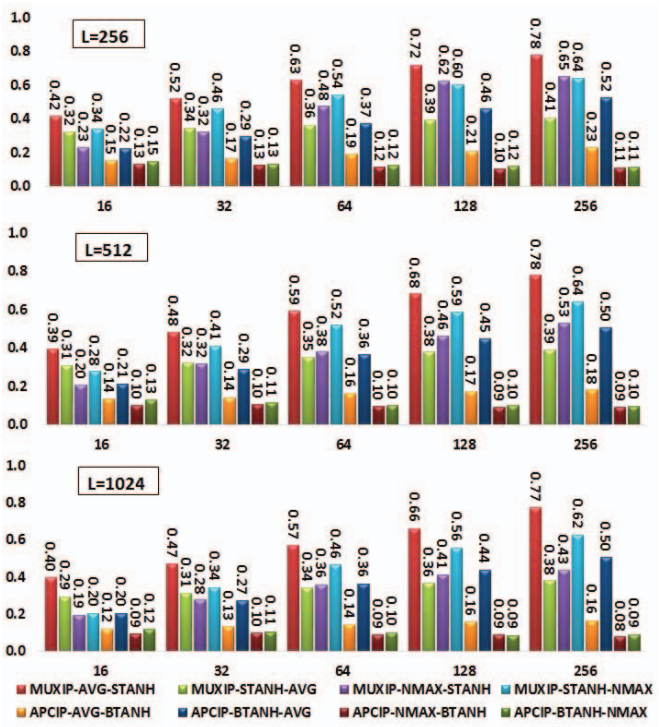

Fig. 2. Imprecision for Optimized FEBs with bit-stream length $L = 256, 512, 1024$

K is the nearest even number of the result calculated by the corresponding function. Please note in No. 5, each set of output of the average pooling block is a binary number instead of stochastic a bit-stream, the formula proposed by [14] to determine the optimal state number should be modified because of the existence of the average pooling block. Thus the parameter α is adjusted to 0.5. In No. 6 and 8, the BTANHs are directly connected with APCIPs just as they are originally designed, thus the formula proposed in [14] to determine the optimal state number is not modified. However, since we do not conduct the pre-scaling, the scaling factor s in [14] is 1, α is adjusted to 2. In No. 7, since the NMAX pooling block for binary numbers is accurate (implemented by accumulators), APCIP is regarded to connect with BTANH directly, then the same parameter of $\alpha = 2$ is used.

C. Results for Feature Extraction Blocks

Fig. 2 shows the imprecisions of eight FEB designs. Each evaluation is given $10,000$ sets of random inputs ranging from -1 to 1 with different input sizes, i.e. 16, 32, 64, 128, 256 and bit-stream lengths, i.e. 256, 512, 1024. The average absolute error is used as the measure of imprecision, which is the mean of the absolute difference between the expected results and the observed results for the same test cases. We observed for each design that (i) as a bit-stream gets longer, the absolute error decreases for the same input size. Because a number can be represented more precisely with longer bit-streams. But the improvements are not significant from the observation of each group of bars in Fig. 2. (ii) With the same length of bit-stream, additional inputs result in increase of imprecision.

From MUXIP-AVG-STANH and MUXIP-STANH-AVG, it is observed that hardware-oriented design, where pooling blocks follows activation function, halves that absolute error at its best effort. Generally, APC-based FEBs are more accurate than MUX-based ones, for APCs precisely count and sum 1s in the input bit-streams. However, as a special case, from MUXIP-STANH-AVG and APCIP-BTANH-AVG, we can observe that when the input size reaches 64 or more, the MUX-based design is more accurate than the APC-based one. For both MUX-based and APC-based designs, near-max pooling designs gives competitive and better results than average pooling designs. Because in our average pooling design, we use multiplexers to randomly select bits from one of the inputs as the average of all inputs while in near-max pooling, our selection of the maximum bit-stream from inputs is based on local statistics. This greedy selection mechanism guarantees the precision of designs using max pooling.

V. Network Optimization for DCNN using SC

We evaluate our designs using LeNet-5 [2] network with $784 - 11520 - 2880 - 3200 - 800 - 500 - 10$ configuration, which is designed to identify MNIST [18] handwritten digits. Validation error is evaluated using partial data from training set while test error is evaluated using different data from training set. We use Synopsys Design Compiler to synthesize the DCNNs with the 45nm Nangate Open Cell Library [19]. The design of stochastic number generators (SNGs) proposed in [20] are adopted in this work to generate the stochastic bit-streams. Please note that we use the same inner-product blocks for the neurons in the fully-connected layer as those in the feature extraction layers, and the structural design co-optimization is applied for inner-product blocks and activation functions in fully-connected layers.

In Table. II, we conclude the software-based DCNNs as the reference on the left side and shows the performance for corresponding proposed hardware-based DCNNs on the right side. There are totally four reference software-based models. AVG-TANH represents a software-based LeNet-5 model with FEBs in which average pooling is followed by an activation function of hyperbolic tangent. Similarly, we named other three models as TANH-AVG, MAX-TANH, TANH-MAX. Correspondingly, each software reference model has two hardware implementations listed on the right side. Each DCNN with one hardware FEB design is evaluated with different lengths of bit-streams, i.e. 256, 512, 1024. Both network accuracies (validation error rate and test error rate) and hardware performance are analyzed. The delay and energy are measured for one run of DCNN inference.

It is observed that the DCNNs using MUX-based inner-product blocks (No. $1, 3, 5, 7$) provide smaller footprints while the DCNNs with APC-based inner-product blocks (No. $2, 4, 6, 8$) achieve better network accuracies and lower powers. However, APC-based designs have longer path delays than MUX-based designs with the same bit-stream length correspondingly, which makes APC-based designs' energy consumptions are much higher. Average pooling based designs (No. $1 - 4$) exploit smaller footprints than max pooling based designs (No. $5 - 8$), for its simplicity in the hardware implementation (multiplexers only). For the same reason, average pooling based designs show a lower power/energy than max pooling based designs. The arrangements of pooling neurons and activation functions reflects two different facts that for some designs like No. 1 and No. 3, hardware-oriented modified designs perform better network accuracy (smaller validation and test error), whereas, for some designs like No. 5 and No. 7, SC implementations on software-based structure provide better accuracy and hardware performance.

We also implemented the DCNNs on consumer-class energy-efficient CPU (Intel® Xeon® E3-1230 v2), GPU

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

No.	Software Configuration	Validation Error (%)	Test Error (%)	Area (mm^2)	Power (W)	Delay (ns)	Energy (μJ)	Harware Configuration	Bit-stream Length	Validation Error (%)	Test Error (%)	Area (mm^2)	Power (W)	Delay (ns)	Energy (μJ)
				CPU				MUXIP-	256	10.54	11.55			332.8	1.1
1				160	41.4	876062	36268.97	AVG-	512	9.80	9.96	6.62	3.3	665.6	2.2
	AVG-TANH	1.41	1.34			GPU		STANH	1024	8.69	8.64			1331.2	4.4
				148	54	39910	2155.14	APCIP-	256	1.72	1.69			1280.0	4.0
2						Binary-ASIC		AVG-	512	1.61	1.54	13.98	3.1	2560.0	7.9
				769.30	587.5	4.2	2.44	BTANH	1024	1.48	1.50			5120.0	15.8
				CPU				MUXIP-	256	7.91	8.01			332.8	2.7
3				160	41.4	1069806	44289.99	STANH-	512	6.86	7.38	11.42	8.1	665.6	5.4
	TANH-AVG	1.01	1.02			GPU		AVG	1024	6.11	6.59			1331.2	10.8
				148	54	40969	2212.32	APCIP-	256	2.16	1.95			1280.0	7.9
4						Binary-ASIC		BTANH-	512	1.64	1.75	28.67	6.2	2560.0	15.8
				4334.64	603.1	4.2	2.03	AVG	1024	1.67	1.60			5120.0	31.5
				CPU				MUXIP-	256	7.92	8.12			332.8	2.1
5				160	41.4	865169	35818.04	NMAX-	512	4.78	4.88	10.12	6.4	665.6	4.3
	MAX-TANH	0.93	0.96			GPU		STANH	1024	3.18	2.96			1331.2	8.5
				148	54	30178	1629.59	APCIP-	256	1.11	1.08			1280.0	7.2
6						Binary-ASIC		NMAX-	512	1.15	1.04	28.29	5.6	2560.0	14.3
				770.81	444.2	5.6	3.48	BTANH	1024	1.02	0.96			5120.0	28.6
				CPU				MUXIP-	256	11.19	11.11			332.8	4.4
7				160	41.4	1059372	43858.03	STANH-	512	7.84	8.18	17.51	13.4	665.6	8.9
	TANH-MAX	0.94	0.96			GPU		NMAX	1024	3.99	4.08			1331.2	17.8
				148	54	40119	2166.46	APCIP-	256	1.11	1.21			1280.0	8.7
8						Binary-ASIC		BTANH-	512	1.05	1.12	34.94	6.8	2560.0	17.5
				1067.88	590.6	5.46	3.9	NMAX	1024	1.02	1.06			5120.0	35.0

(Nvidia® GeForce GTX 750 Ti), and synthesized binary computing based *application-specific integrated circuits (ASICs)* (in short, *Binary-ASIC*). The proposed SC-based hardware designs of DCNN are much more area efficient, with improvements up to $24.17\times$, $22.36\times$, and $776.61\times$ compared to CPU, GPU, and synthesized Binary-ASICs respectively. Besides, compared with CPU and GPU, proposed designs of SC-based DCNN outperform in the aspects of energy efficiency and power efficiency. They achieve up to $13.41\times$ power and $32,835.32\times$ energy improvements over CPU implementations as well as up to $17.49\times$ power and $1,951.11\times$ energy improvements over GPU implementations. Regarding the synthesized Binary-ASICs, the proposed designs provide as high as $190.27\times$ power efficiency improvements and up to $2.21\times$ energy efficiency improvement. Please note in each Binary-ASIC synthesis, we implemented an ideal full-parallel pipelined structure where SC-based components are replaced with binary-based components and we used 8-bit fix-point numbers for the implementation. Thus the Binary-ASICs conducted the inference of a DCNN much faster, which made the energy efficiency improvement by SC-based designs not significant. But in reality, the power ($400 \sim 600W$) and area ($700 \sim 4000mm^2$) of the Binary-ASIC syntheses are not acceptable. With the sequential logic to reduce the area and power, the delay and energy consumption of the Binary-ASICs will be considerably increased, so that the SC-based DCNNs have the potential to achieve more substantial energy improvements. The proposed SC-based DCNNs can achieve as low as 1.02% validation error rate (No. 8) and 0.96% test error rate (No. 6) which are extremely close to corresponding software-based model results. In those cases, the power, area, and energy of proposed SC-based designs are very small compared to CPU and GPU implementations. Meanwhile, those SC-based designs are more power/area efficient than synthesized Binary-ASICs.

REFERENCES

[1] 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.

[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] 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.

[4] 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.

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

[6] 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.

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

[8] A. Ren, J. Li, Z. Li, C. Ding, X. Qian, Q. Qiu, B. Yuan, and Y. Wang, "Sc-dcnn: highly-scalable deep convolutional neural network using stochastic computing," *arXiv preprint arXiv:1611.05939*.

[9] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[10] N. Y. Hammerla, S. Halloran, and T. Ploetz, "Deep, convolutional, and recurrent models for human activity recognition using wearables," *arXiv preprint arXiv:1604.08880*, 2016.

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

[12] 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.

[13] 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.

[14] 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.

[15] Y. Ji, F. Ran, C. Ma, and D. J. Lilja, "A hardware implementation of a radial basis function neural network using stochastic logic," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 880–883.

[16] 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.

[17] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016.

[18] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.

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

[20] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 256–261.