Scalable Stochastic-Computing Accelerator for Convolutional Neural Networks

Hyeonuk Sim*, Dong Nguyen*, Jongeun Lee*, [1], Kiyoung Choi†
*School of Electrical and Computer Engineering, UNIST, Ulsan, South Korea
{detective, dongnguyen, jlee}@unist.ac.kr
†Dept. of Electrical and Computer Engineering, Seoul National University, Seoul, South Korea
kchoi@dal.snu.ac.kr

***Abstract*—Stochastic Computing (SC) is an alternative design paradigm particularly useful for applications where cost is critical. SC has been applied to neural networks, as neural networks are known for their high computational complexity. However previous work in this area has critical limitations such as the *fully-parallel architecture* assumption, which prevent them from being applicable to recent ones such as convolutional neural networks, or *ConvNets*. This paper presents the first SC architecture for ConvNets, shows its feasibility, with detailed analyses of implementation overheads. Our SC-ConvNet is a hybrid between SC and conventional binary design, which is a marked difference from earlier SC-based neural networks. Though this might seem like a compromise, it is a novel feature driven by the need to support modern ConvNets at scale, which commonly have many, large layers. Our proposed architecture also features hybrid layer composition, which helps achieve very high recognition accuracy. Our detailed evaluation results involving functional simulation and RTL synthesis suggest that SC-ConvNets are indeed competitive with conventional binary designs, even without considering inherent error resilience of SC.**

***Keywords*—Convolutional neural network, Stochastic computing, Recognition accuracy, Weight parameter retraining.**

I. Introduction

Stochastic Computing (SC) is an alternative design paradigm where numbers are represented by the frequency of ones in a bitstream, also called *signal probability*. The bitstream itself may be arranged serially, parallelly, or in some combination of the two, but unlike in conventional binary (abbreviated as CB) number systems, there are no positional weights among bits, or all positions have the equal weight. This property gives SC circuits unique strengths in terms of (i) better error resilience and (ii) free, dynamic tradeoff between performance, accuracy, and energy, in addition to the fact that it allows for (iii) extremely cheap implementations of complex arithmetic operations (e.g., multiplication performed by one NAND or XNOR gate). The downsides include inherent fluctuation of output and long latency.

Neural networks have very high computational complexity and high error-tolerance at the algorithmic level, which make them a good candidate for SC-based acceleration. In particular convolutional neural networks also known as *ConvNets* can provide the highest and very robust recognition performance across a wide spectrum of visual recognition tasks, and are very popular despite their demanding computational complexity. A SC-based ConvNet that is low in cost but which can at times boost its accuracy when high energy is affordable would be highly desirable.

Previous work on SC-based neural networks has shown the viability of SC in the context of neural networks, and proposed ideas on overcoming certain limitations of SC [1]–[3]. However previous work has some critical limitations, such as the *fully-parallel architecture* assumption and using *fully-connected networks* only. This prevents the previous solutions from being applicable to recent ConvNets.

There are at least three problems in applying previous SC-based neural-net solutions to ConvNets. First, ConvNets typically consist of many layers, with GoogLeNet having more than a hundred layers. It is virtually impossible to implement all the layers in parallel using dedicated hardware on a single chip even in SC. Second, ConvNets have many layer types whose SC versions are unknown. Examples include max-pooling layer (due to the difficulty of doing comparison in SC), ReLU (Rectified Linear Unit) layer, and local response normalization (LRN) layer. Third, even if one solves the first two problems, it remains uncertain whether the accuracy will be high enough. No one has shown that SC-ConvNet can achieve even similar recognition performance as CB (Conventional Binary) designs. This is a very critical issue, as low-performing neural network has very little value.

As a first step towards evaluating the feasibility of SC for ConvNets, in this paper we consider a hybrid design where only part of the architecture is in SC. Clearly this marks an important departure from earlier work on SC-based neural networks, but it is prompted by the unique characteristics of ConvNets and the need to support ConvNets at scale.

In this paper we present a scalable SC-based accelerator for ConvNets targeting low-cost embedded applications. We make the following contributions in this paper.

1) We argue that for scalability, it is essential to have some elements designed in conventional binary representation. Based on this, we present a scalable SC-based accelerator architecture for ConvNets.
2) To enhance accuracy of SC-ConvNets, we propose a technique called *hybrid layer composition*, one form of which is to use SC for the convolution layers only while using conventional binary for fully-connected layers.
3) Through our evaluations involving functional simulation and RTL synthesis, we demonstrate that SC is a viable option for ConvNets, and that SC-ConvNets are competitive with conventional binary designs, even without considering advantages of SC in physical implementation and dynamic energy-quality tradeoff.

II. Related Work

Many techniques have been proposed to address the high computational complexity of deep neural networks DNNs (Deep Neural Networks). At the algorithm level, researchers

[1]Corresponding author

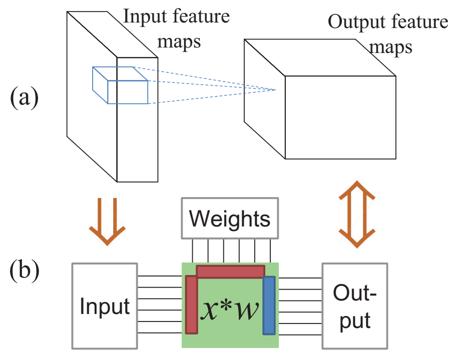


Fig. 1: A generic accelerator for convolution layers.

have proposed model reduction such as low-rank approximation, which can reduce complexity with little impact on accuracy. It is also shown that storage requirement due to weight parameters can be reduced to its 5 to 10% level [4], which may alleviate memory bottleneck problem. However these techniques are effective mainly for fully-connected layers, with much less success with convolution layers, which account for the majority of computational complexity. At the implementation level, reducing the precision of arithmetic operations can reduce hardware cost, without significantly affecting accuracy. Approximate computing has also been applied to *some* of the neurons [5], which may be exploited to improve energy efficiency of neural-net accelerators.

All these results strongly suggest that current DNN models are highly redundant, and essentially the same recognition performance can be achieved through retraining with much less weight parameters and less precise computation. That is why we may expect higher energy efficiency through SC, which is known to be more efficient for low-precision computation.

With limited resources, not all neurons can be simultaneously active, causing resources to be shared among different neurons. Hence using a good architecture can make a difference in terms of performance and energy efficiency. Several architectures for neural-net acceleration have been proposed in the domain of conventional binary computing as well [6]–[9].

As early as in 1994, SC was applied to implementing neural networks on a then resource-limited FPGA [10]. Circuit elements for SC are described in [11], with a companion paper [12] giving a neural net design as an example. Recently SC has been applied in designing Deep Belief Networks [2], [13] and other non-ConvNet-type fully-connected neural networks [3], [14]. We target ConvNets, which are more widely used today but much more complicated than the aforementioned fully-connected types of neural networks.

The StoRM approach [15] is similar to ours in that both focuses on a MAC array. However our proposed techniques are tied to ConvNets, and designed to improve scalability and accuracy of SC-ConvNet accelerators whereas [15] is not.

III. Preliminary: ConvNet Acceleration

Let us consider one convolution layer as convolution layers in a typical ConvNet account for more than 90% of the MAC (multiply-accumulate) operations. The computation in a convolution layer can be seen as a transformation from one 3D array of data called *input feature maps* to another 3D array called *output feature maps*, as illustrated in Figure 1a. Each element of the output array is computed by the 3D convolution between the input array and the weight matrix, followed by a bias addition. Different output array elements are created by applying the same convolution operation for different input subarrays or different weight subarrays.

One important point here is that the input/output feature maps can be very large, and is stored in the off-chip memory in the general case. The off-chip memory may have sufficiently large capacity, but accessing it requires memory bandwidth, which is often quite limited. Another factor is on-chip buffer size. In general there can be three on-chip buffers, for input, output, and weight parameters (see Figure 1b), and unless the entire data can fit in the buffer, buffers need to be double-buffered to hide the off-chip memory access latency. Since weight parameters for convolution layers are much smaller than input/output data, we assume in the following that weight parameters completely fit in an on-chip buffer, but input/output buffers are large only enough to supply input/output data without ceasing through double buffering.

The computation itself is very simple. We need no operation but a number of MAC operations. Accelerators are often built around an array of hardware MAC units, which can be organized as simply as a 1D array or a 3D array. The exact shape (such as 1D vs. 2D) and the size of the MAC array, along with the hyper-parameters of each layer (number of neurons, size of filters, etc.), affect the performance of the accelerator.

IV. Tile-Parallel Binary-Interlaced SC Architecture

A. Scalability Requires Tile-Parallel Architecture

Previous work on SC-neural nets [2], [3], [13] has assumed that every layer and neuron can have dedicated resources. At the same time, since the networks they use are all fully-connected, it means that the application throughput is one (e.g., 1 image per cycle), and there is no need to buffer any intermediate result for any extended length of time. This is an ideal situation in terms of energy efficiency, since no memory and no resource sharing mean almost no power dissipation for anything but computation. Such a fully-parallel architecture may be realizable for special applications where the area budget is large and the size of the neural network is small. It may also be how biological neural networks operate.

However for low-cost embedded applications and/or if targeting real-life ConvNets, the fully-parallel architecture quickly becomes very irrelevant. We need an architecture that can support large ConvNets as well. A typical solution is to partition *computation* into multiple equal-sized blocks (or *tiles*) in a manner analogous to loop tiling, and to design hardware for one computation tile. Such hardware requires on-chip buffers for fast processing, but on-chip memory is invariably too small to hold entire data even for one layer. Double buffering is commonly employed, where on-chip buffers hold only the portion of the data currently needed while the entire layer data are stored in the off-chip memory. The on-chip buffers should be large enough to hide the data transfer latency. In this paper we call such an architecture *tile-parallel architecture*.

It is easy to see that tile-parallel architectures are a necessary condition for scalability—that is, for an accelerator to support ConvNets of many, large layers. Tile-parallel architecture, in turn, implies that stochastic computation must be interlaced with conventional binary storage of data, in order to avoid a high penalty in terms of on-chip memory and off-chip bandwidth. We call this *binary-interlaced SC* design. The alternative is to use fully-stochastic design, which can avoid the overhead of doing multiple conversions between SC and CB (Conventional Binary).

B. Rationale for Binary-Interlaced SC Design

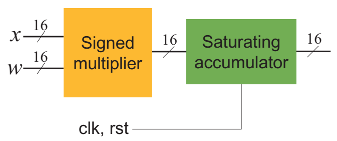
Compared with CB, SC uses much less resources for computation but requires far more cycles to achieve an equivalent resolution. The simplicity of SC circuits can help achieve higher clock speed. Let us assume that SC needs 2^k bits to get a similar resolution of k-bit fixed-point implementation in the conventional binary. One can see that from the throughput point of view, SC can beat conventional binary when k is low.

When it comes to handling data, such as storing them on-chip or sending them to off-chip, conventional binary is more efficient for *any value* of k, although a conventional binary representation may also be more susceptible to errors. Thus fully-stochastic design such as in [2], [3] is largely incompatible with the idea of tile-parallel architecture. Otherwise the overhead in on-chip memory capacity and off-chip bandwidth can be many-fold: as much as 32$\times$ for $k = 8$, and 4$\times$ even when $k = 4$. Further, given that a large portion of the die area is dedicated for on-chip memory rather than the MAC array itself [8], the many-times increase in overhead in the case of fully-stochastic design seems simply too high. Thus we conclude that for scalable architecture it is necessary to have binary-interlaced design. We assume that the conversion from stochastic bitstreams to binary numbers happens before they are saved to on-chip memory.

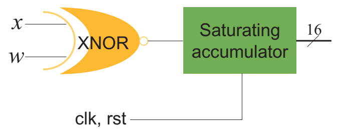
C. Our Proposed Architecture: Doubly Hybrid

1) Binary-Interlaced SC for Scalability: Our ConvNet accelerator uses SC for the MAC array only, converting the input and weights from conventional binary to SC bitstreams, and converting back the output bitstreams to binary. Coincidentally this helps us avoid implementing other layers in SC, such as max-pooling layer, ReLU layer, and LRN layer, which can be tricky to implement in SC. We perform even the accumulation (of a MAC operation) using conventional binary arithmetic, utilizing SC for multiplication only. Multiplication is quite important, accounting for 76% of the area in our synthesis result for a conventional-binary MAC design (see Figure 3a). Thus even with this hybrid approach we can expect some efficiency improvement.

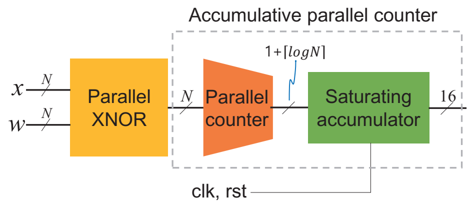
Perhaps one of the biggest downsides of this scheme is the conversion overhead between SC and conventional binary. It is not that there are unnecessarily many converters all around the accelerator; physically, there is only one array of binary-to-stochastic (B-to-S) converters and one array of stochastic-to-binary (S-to-B) converter, as shown in red/blue blocks in Figure 1b. But these converters are used for every layer of a ConvNet, which may seem redundant and unnecessary from the viewpoint of a fully-parallel, fully-stochastic-computing accelerator. However, we see this as a fair price to pay, necessary to support large ConvNets on a small accelerator. Also the B-to-S overhead can be amortized by sharing resources such as LFSRs (Linear Feedback Shift Registers) across the MAC array, or reduced significantly by using emerging devices such as magnetic tunnel junction (MTJ) transistors [16]. We address the S-to-B overhead by using approximation and bit-parallel SC design (see Section IV-D).



(a) Conventional binary MAC



(b) 1-bit SC-MAC



(c) N-bit parallel SC-MAC

Fig. 2: Structure of a MAC unit. Our SC-MAC integrates a stochastic-to-binary (S-to-B) converter (i.e., parallel counter).

2) Hybrid Layer Composition for Accuracy: The other feature of our architecture is that SC is applied to convolution layers only whereas fully-connected layers use conventional binary, which is one form of what we term *hybrid layer composition*. This is the rationale. The number of MACs in the fully-connected layers of a ConvNet is usually very small, but just because they are very close to the output, they can have a direct impact on the final outcome. Using conventional binary for the fully-connected layers can therefore ensure high accuracy, while using SC for convolutional layers can increase cost-effectiveness, thus striking a good balance between accuracy and efficiency. Though simple, it does not cost any more hardware and works surprisingly well.

D. SC-MAC Unit: Minimizing Overhead with Parallelism and Approximation

Figure 2 illustrates our SC-MAC design in comparison with a conventional binary one. In SC, multiplication and addition can be performed very cheaply by an XNOR gate and a MUX, respectively [11]. The problem is that they are too small, compared with the ensuing S-to-B converter, which is typically done by a counter. This counter can also serve as the accumulator, leading to a very simple 1-bit SC-MAC design as shown in Figure 2b. But in this design the overhead of the accumulator (also serving as a counter) can be nearly 100 times (see Figure 3b), in terms of area according to our evaluation.

We address this problem by employing bit-parallel SC design and approximation. An N-bit parallel SC design allows us to share the accumulator among N 1-bit SC-MACs, essentially reducing the accumulator overhead by N times. However, in a bit-parallel version we need an explicit counter, such as an N-bit parallel counter, as illustrated in Figure 2c. Since the size of the accelerator is largely unchanged regardless of N, we can reduce the accumulator overhead very effectively with bit-parallelism. The parallel counter overhead, however, is a different story, which we tackle using approximation. Nonetheless by putting together a parallel counter and an accumulator, which are together known as *accumulative parallel counter*, we can exploit known circuit optimization techniques such as [17] to achieve higher operating frequency and smaller area.

The parallel counter overhead cannot be reduced by going parallel, because it increases in proportion to N. Instead we use an approximate S-to-B method [18]. The idea is to reduce N bits into $N/2$ bits with approximately half the number of "one"s. While this can nearly halve the parallel counter overhead, it has a side effect—lowering the recognition accuracy, as a result of which we may end up using a longer bitstream, defeating the purpose of using approximation. Thus we need a careful evaluation to assess its effectiveness (see Section V).

V. Experiments

A. Experimental Setup and Implementation Detail

An evaluation of any SC design must consider both functional accuracy and the efficiency of hardware implementation. In the case of SC-ConvNet, we also need to consider (re)training of weight parameters, which is really crucial as we will show. Fortunately for us, we were able to easily extend an existing ConvNet model, Caffe [19], to generate a model of our SC-ConvNet and use it for training and functional simulation, because our architecture retains the same interface as the conventional binary version.

To see the efficiency of hardware implementation we also designed and implemented our SC-MAC unit. We designed not just one SC-MAC unit, but an array of SC-MAC units, which we call *compute tile* or simply a *tile*, as needed in a ConvNet accelerator. Again, because our SC-MAC compute tile has the same interface as its binary version, we can easily and accurately evaluate our proposal without necessarily building the entire system in hardware.

We use a ConvNet designed for the MNIST digit recognition benchmark, which has been frequently used in previous work [2], [3], [13]. Ours is based on a ConvNet model distributed with Caffe, but modified by adding *tanh*-activation layers, in line with previous work using the same dataset.

1) MAC Design: We have implemented different MAC designs in Figure 2 using Verilog RTL. For bit-parallelism N, we use 64 and 128 (The approximate version is 128-bit only). In the binary MAC, we use fixed-point consisting of a 1-bit sign bit and a 15-bit fractional part. No integer part is necessary as values are between -1 and 1. The multiplier generates 16-bit signed number. The accumulator, which is shared with SC-MACs, uses 18 bits internally due to a 2-bit integer part, but the final output is given by the most significant 16 bits. The parallel counter is an up/down counter as our stochastic bitstream follows a bipolar representation.

For accurate area evaluation we have implemented the entire compute tile, including LFSR-based B-to-S, referred to as SNG (Stochastic Number Generator), and an MAC array, whose size is varied from 4x4x4 to 8x8x8. For synthesis we use Synopsys

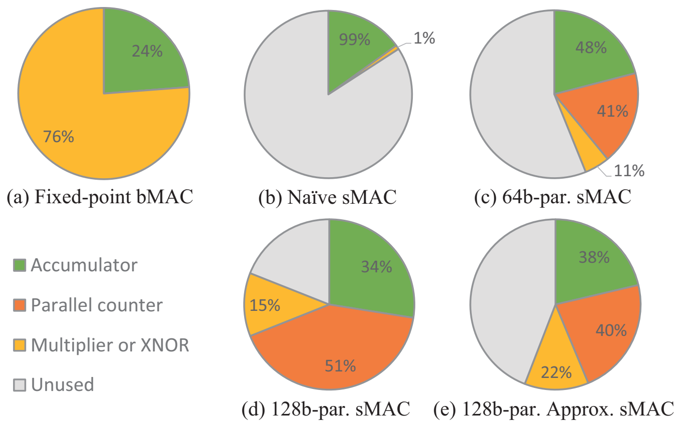


Fig. 3: Area breakdown of a MAC, where bMAC and sMAC stand for a conventional binary MAC and an SC-MAC, respectively. Each whole pie, including the *unused* portion shown in gray, represents the same area, which is the area of the bMAC; sMAC uses far less. A number in each pie represents the portion of a component in the *used* area.

Design Compiler with TSMC 45nm technology. The target clock period is set to 0.65ns from the best achievable value by the binary MAC array. No pipelining is used inside a MAC unit.

2) Simulation and Training: We have extended Caffe with a new convolution layer definition and a new fully-connected layer definition based on our SC-MAC array. The binary input and weight parameters are first converted into stochastic bitstreams and go through the SC-MAC array, after which the binary-converted output is generated. The binary input values are within -1 and 1 due to *tanh* function, but weight values may not be. Thus we scale them before convolution and re-scale back after convolution. We use 4 and 2 for scale parameters for the first and second convolution layers, respectively.

Due to the way the convolution operation is defined mathematically, the input data are reused for computing different output feature maps. Our modeling, as well as our hardware architecture, assumes that these input data are reused at the binary level as opposed to the stochastic bistream level, and new stochastic bitstreams are generated from the same binary input data whenever they are needed. Retraining of SC-ConvNet is done by applying SC to the forward pass only. The B-to-S conversion is done by comparing a random number with the binary input. For random number generation we tested both LFSR and CUDA random function, but the latter, which is used in all our results, turned out to be orders-of-magnitude faster than LFSR without much difference in accuracy.

B. Hardware Implementation Efficiency

Figure 3 shows the area breakdown of a MAC unit. The area of a MAC (without the gray portion in the figure) is the average value obtained by dividing the tile area by the number of MAC units. The figure shows that the accumulator area is almost the same for all the cases, and the parallel counter area is proportional to the bit-parallelism. The approximate version is very effective in reducing the parallel counter overhead.

TABLE I: Compute tile synthesis results

Tile size	Case	Area (mm^2) SNG	Area (mm^2) MAC array	Peak Performance
4x4x4	Binary	—	0.094	98.5G MAC/s
	64b-SC	0.074	0.042	6.3T sMAC/s
	128b-SC approx.	0.151	0.053	12.6T sMAC/s
8x8x8	Binary	—	0.765	787.7G MAC/s
	64b-SC	0.220	0.336	50.4T sMAC/s
	128b-SC approx.	0.448	0.426	100.8T sMAC/s
12x12x12 (est.)	Binary	—	1.494	2.7T MAC/s
	64b-SC	0.442	1.134	170.1T sMAC/s
	128b-SC approx.	0.904	1.438	340.3T sMAC/s

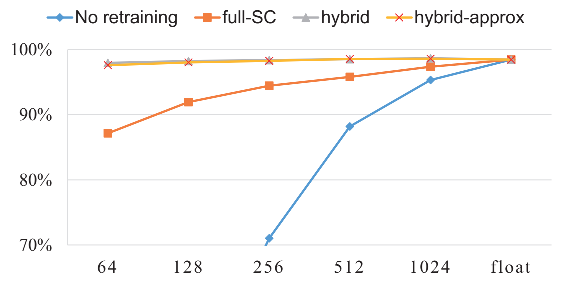


Fig. 4: Recognition accuracy vs. stochastic bitstream length.

Under the same area-delay product, naïve SC-MAC (1-bit serial version) can process 8 bits while the binary version processes 1 word. Our 128-bit approximate version can process nearly 256 bits, which is about 30X improvement. Increasing bit-parallelism increases parallel counter area and latency, but does not increase the overall MAC latency because the parallel counter and the accumulator can be optimized together as an accumulative parallel counter. The larger area of the accumulator in Figure 3d is due to the increased critical paths, which increase the areas of both the parallel counter and accumulator due to timing optimization.

Table I summarizes tile synthesis result, where the data for tile size of 12x12x12 are extrapolated (could not be synthesized due to size). Extrapolation is possible because the area is linearly proportional to the size of the array and increasing tile size does not affect the critical path (affected most by bit parallelism). The peak performance is the theoretical performance assuming that the MAC units are utilized 100%. In practice, utilization varies depending on the layer parameters and whether there other bottlenecks in the system such as data transfer. For the SC versions we use sMAC/s, which is the number of 1-bit SC-MAC operations performed per sec., since the application throughput depends on the stochastic bitstream length. The table shows that the SNG overhead, which includes that of both input and weight parameters, is quite high when the tile is small, but reduced quickly as the tile size increases.

C. Overall Performance

Figure 4 shows the recognition accuracy of different schemes for different Stochastic Bitstream Length (SBL) values. The original ConvNet using floating-point arithmetic is first trained using the default training script provided in Caffe, which runs 10,000 iterations. It is followed by retraining of 5,000 iterations using different versions of SC-ConvNet. We use the same parameters except that the base learning rate is decreased by 10 times, which is a common practice.

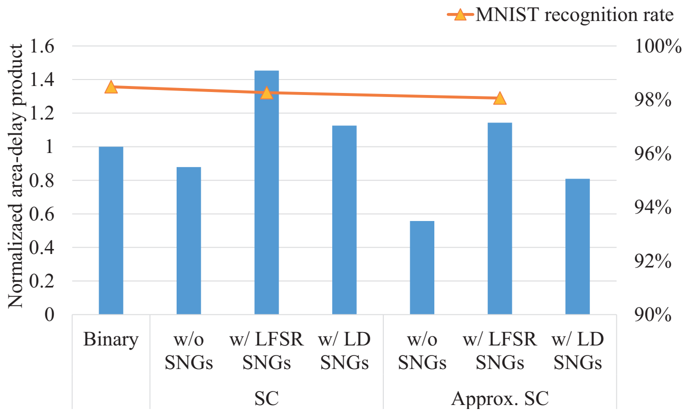


Fig. 5: Area-delay product comparison (SBL: 128-bit).

These graphs are obtained after retraining the network, except for the floating-point case. Without retraining the recognition accuracy is extremely poor, being around 88% even for SBL of 512 bits (the full-SC ConvNet case), with steep decline for lower values of SBL.

The graph clearly shows that there is a significant gap in recognition performance between our hybrid ConvNet and full-SC ConvNet (i.e., fully-connect layers are in SC as well), suggesting the superiority of the hybrid design. Between the two versions of hybrid design—approximate vs. non-approximate versions—we see that the difference in recognition performance is negligible. However the MNIST dataset is relatively easy, and we may find larger gap for more complex dataset, which is left for future work.

Figure 5 shows the overall performance as measured in area-delay product, which is the inverse of throughput per area, for different cases. For this comparison we use SBL of 128 bits, for both approximate and non-approximate SC-MAC cases, and the tile size is 8x8x8. Tile size being the same, utilization must also be the same for all the cases considered here, regardless of layer parameters.

The graph shows that in terms of the recognition accuracy, there is only less than 1% point difference among them. Yet in terms of throughput the approximate version can achieve about 35% higher performance over the non-approximate version, without considering SNGs. If SNGs are not considered, the best SC version can be nearly twice as area-efficient as the binary version. With recent MTJ devices the cost of SNG is getting much cheaper, but here due to the lack of accurate area information of MTJ-based random number generators, we instead use the area of the recent low-discrepancy (LD) SNGs [20], which are designed for bit-parallel SC circuits and therefore well-suited for our application. This can reduce the area overhead of SNGs down to about 50% of the MAC array. Overall this graph suggests that SC-ConvNet can be both accurate and area-efficient, even with SNG overhead, compared with a conventional binary version. Note that this comparison is without considering the inherent advantages of SC such as dynamic energy-quality tradeoff and better error tolerance. For future technologies in which variability and noise are expected to grow, the advantages of SC will be greater.

Finally Table II compares our proposed SC-ConvNet archi-

TABLE II: Comparison with previous neural-net accelerators

		Frequency	Area*	Power*	GOPS	GOPS/mm^2	GOPS/W	Tech.	Scope for area & power
Binary	MWSCAS'12 [6]	400 MHz	12.50	570.00	160.00	12.80	280.70	45nm	Total chip
	ISSCC'15 [7]	200 MHz	10.00	213.10	411.30	41.13	1930.08	65nm	Total chip
	ASPLOS'14 [8]	980 MHz	0.85	132.00	501.96	592.94	3802.73	65nm	NFU** only
	GLSVLSI'15 [9]	700 MHz	0.98	236.59	274.00	278.85	1158.11	65nm	SoP ($\simeq$ MAC) units only
SC	Arxiv'15 [13]	400 MHz	0.09	14.90	1.01	11.91	67.93	65nm	One neuron
	DAC'15 [3]	1000 MHz	0.06	3.60	75.74	1262.33	21038.79	45nm	One neuron with 200 inputs
	Proposed (128b-SC approx.)	1540 MHz	0.43	279.31	1575.38	3697.81	5640.23	45nm	MAC array (size: 8x8x8)

*Note 1: For the scope for area (in mm^2) and power (in mW), see the rightmost column. **Note 2: NFU can also perform pooling and activation function.

tecture's performance with previous neural-net accelerators. We use 128-bit approximate version with the tile size of 8x8x8. Due to the differences in many aspects including the target neural network, we compare performance in GOPS (Giga operations per sec.), with 1 MAC being equivalent to 2 operations. In all SC cases, SNGs are excluded from area and power calculation. Note also that the first two cases (MWSCAS'12 and ISSCC'14) are not directly comparable with the rest, since they include large on-chip buffers, which should dominate in any neural network accelerator.

Compared to a recent SC design [3], ours has much higher area efficiency but dissipates much more power, which is due to the aggressive clock speed optimization and the accumulator's power overhead in ours. The previous work [3] is essentially a fully-parallel architecture, which is one reason why it has supreme energy efficiency. Instead ours have scalability, which cannot be provided by [3]. Compared to the others, our architecture is actually quite energy-efficient in addition to having the highest area-efficiency.

VI. Conclusion

In this paper we presented SC-based ConvNet accelerator, in which the majority of computation is done in SC. By combining conventional binary alongside SC, an entire ConvNet can be easily evaluated and implemented, in addition to ensuring scalability of the architecture. To improve the accuracy of SC-ConvNet, we proposed a hybrid SC-binary layer architecture. To minimize the overhead of SC-binary conversion, we proposed a parallel and approximate SC-MAC. Our detailed evaluation demonstrate that SC-ConvNets are indeed viable and can be competitive with conventional binary designs.

Evaluation and training of SC-ConvNets takes orders-of-magnitude longer runtime even when using state-of-the-art GPUs, which is why the size of the ConvNets we use in this work is rather limited. We plan to address the speed issue and evaluate our techniques on larger SC-ConvNets.

Acknowledgments

This work was supported in part by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-IT1501-08.

References

[1] V. Canals *et al.*, "A new stochastic computing methodology for efficient neural network implementation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 551–564, March 2016.

[2] K. Sanni *et al.*, "Fpga implementation of a deep belief network architecture for character recognition using stochastic computation," in *Information Sciences and Systems (CISS), 2015 49th Annual Conference on*, March 2015, pp. 1–5.

[3] K. Kim *et al.*, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 124:1–124:6.

[4] S. Han *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[5] Q. Zhang *et al.*, "Approxann: An approximate computing framework for artificial neural network," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 701–706.

[6] P. H. Pham *et al.*, "Neuflow: Dataflow vision processing system-on-a-chip," in *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2012, pp. 1044–1047.

[7] S. Park *et al.*, "4.6 a1.93tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications," in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, Feb 2015, pp. 1–3.

[8] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 269–284.

[9] L. Cavigelli *et al.*, "Origami: A convolutional network accelerator," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15. New York, NY, USA: ACM, 2015, pp. 199–204.

[10] S. L. Bade *et al.*, "Fpga-based stochastic neural networks-implementation," in *FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on*, Apr 1994, pp. 189–198.

[11] B. Brown *et al.*, "Stochastic neural computation. i. computational elements," *Computers, IEEE Transactions on*, vol. 50, no. 9, pp. 891–905, Sep 2001.

[12] ——, "Stochastic neural computation. ii. soft competitive learning," *Computers, IEEE Transactions on*, vol. 50, no. 9, pp. 906–920, Sep 2001.

[13] A. Ardakani *et al.*, "VLSI implementation of deep neural network using integral stochastic computing," *CoRR*, vol. abs/1509.08972, 2015.

[14] Y. Ji *et al.*, "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*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 880–883.

[15] V. K. Chippa *et al.*, "Storm: A stochastic recognition and mining processor," in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ser. ISLPED '14. New York, NY, USA: ACM, 2014, pp. 39–44.

[16] R. Venkatesan *et al.*, "Spintastic: Spin-based stochastic logic for energy-efficient computing," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1575–1578.

[17] B. Parhami *et al.*, "Accumulative parallel counters," in *Proc. 29th Asilomar Conf. Signals, Systems, and Computers*, 1995, pp. 966–970.

[18] K. Kim *et al.*, "Approximate de-randomizer for stochastic circuits," in *2015 International SoC Design Conference (ISOCC)*, Nov 2015, pp. 123–124.

[19] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

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