SkippyNN: An Embedded Stochastic-Computing Accelerator for Convolutional Neural Networks

Reza Hojabr[1], Kamyar Givaki[1], SM. Reza Tayaranian[1], Parsa Esfahanian[2],
Ahmad Khonsari[12], Dara Rahmati[2], M. Hassan Najafi[3]
[1]School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran
[2]School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
[3]School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA
{r.hojabr,givakik,m.taiaranian}@ut.ac.ir,{parsa.esfahanian,ak,dara.rahmati}@ipm.ir,najafi@louisiana.edu

ABSTRACT

Employing convolutional neural networks (CNNs) in embedded devices seeks novel low-cost and energy efficient CNN accelerators. Stochastic computing (SC) is a promising low-cost alternative to conventional binary implementations of CNNs. Despite the low-cost advantage, SC-based arithmetic units suffer from prohibitive execution time due to processing long bit-streams. In particular, multiplication as the main operation in convolution computation, is an extremely time-consuming operation which hampers employing SC methods in designing embedded CNNs.

In this work, we propose a novel architecture, called SkippyNN, that reduces the computation time of SC-based multiplications in the convolutional layers of CNNs. Each convolution in a CNN is composed of numerous multiplications where each input value is multiplied by a weight vector. Producing the result of the first multiplication, the following multiplications can be performed by multiplying the input and the differences of the successive weights. Leveraging this property, we develop a differential Multiply-and-Accumulate unit, called *DMAC*, to reduce the time consumed by convolutions in SkippyNN. We evaluate the efficiency of SkippyNN using four modern CNNs. On average, SkippyNN offers 1.2x speedup and 2.7x energy saving compared to the binary implementation of CNN accelerators.

1 INTRODUCTION

A wide range of applications based on convolutional neural networks (CNNs) are emerging in various areas of computer vision. In particular, employing CNNs in intelligent embedded devices interacting with real-world environment has led to the advent of efficient CNN accelerators. Limited computation resources and inadequate power budget are two important challenges when applying neural networks to embedded devices. Customized hardware implementations have gained a lot of attention in recent years to tackle these challenges [5, 6, 25].

Recently, a handful of works have exploited stochastic computing (SC) [2] in designing low-cost CNN accelerators [4, 4, 7, 11, 13, 14, 16, 19, 22, 23]. Compared to the conventional binary implementations, SC-based implementations often offer a lower power consumption, a lower hardware area footprint, and a higher tolerance to soft errors (i.e. bit flips) [3]. In SC, each number X (that is interpreted as the probability $P(x)$ in range [0,1]), is represented by a bit-stream in which the density of 1s denotes $P(x)$ [3]. For instance, a binary number $X = 0.101_2$ that is interpreted as $P(x) = 5/8$, can be represented by a bit-stream $S = 11101001$ where the number of 1s appeared in the bit-stream and the length of the bit-stream are five and eight, respectively. Bit-stream-based representation makes SC numbers more tolerable to the soft errors compared to conventional binary radix representation. A single bit-flip in binary representation (e.g., a bit-flip in the most significant bit) may lead to a huge error while in a SC bit-stream can cause only a small change in the value. Simplicity of design is another important advantage. Most arithmetic operations require extremely simple logic in SC. For instance, multiplication operation is performed using a single AND gate which has a considerably lower hardware cost than the binary multiplier [2, 15].

Despite these benefits, SC-based operations encounter two important problems: low accuracy and long computation time [2]. Prior work showed that due to the approximate nature of neural networks, CNN accelerators can be implemented by low-bitwidth binary arithmetic units at no accuracy loss [6, 21, 26]. Our observations further confirm that, similar to binary implementations, with long enough bit-streams SC-based units do not impose a considerable degradation on the neural network accuracy. Nevertheless, there is still a great demand to decrease the computation time and to improve the energy efficiency of SC-based CNN accelerators.

In this work, we propose a novel SC-based architecture, SkippyNN, which aims at reducing the computation time of stochastic multiplications in the convolution kernel as these operations constitute a substantial portion of computation load in modern CNNs. Each convolution is composed of numerous multiplications where an input x_i is multiplied by the successive weights $w_1, ..., w_k$. Computation time of SC-based multiplications is proportional to the bit-stream length of the operands. Provided by maintaining the result of $(x_i \times w_1)$, to calculate the term $x_i \times w_2$, we can calculate $x_i \times (w_2 - w_1)$ and then add the result to $x_i \times w_1$ which is already prepared. Employing this arithmetic property results in a considerable reduction in the multiplication time as the length of $w_2 - w_1$ bit-stream is less than the length of w_2 bit-stream in the developed architecture. We introduce a differential Multiply-and-Accumulate unit, called DMAC, to exploit this property in the SkippyNN architecture. By sorting the weights in a weight vector, SkippyNN minimizes the differences between successive weights and consequently, minimizes the computation time of multiplications.

In convolutional layers, each filter consists of both positive and negative weights. The conventional approach to handle signed operations in the SC-based designs is by using the bipolar SC domain [2, 19]. The range of numbers is extended from [0, 1] in the unipolar domain to [-1, 1] in the bipolar domain at the cost of doubling the length of bit-streams and so doubling the processing time.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317911

Authors in [11] and [12] proposed to divide the weights into negative and positive subsets and employ unipolar SC operations for each subset instead of using bipolar SC. Although employing this approach eradicates the cost of bipolar SC operations, it requires duplicating most parts of operational circuits such as multipliers and adders. Since the differences of the sorted successive weights are always greater than zero, the weight buffer in SkippyNN consists of only positive numbers. Therefore, SkippyNN eliminates the need for separating the computations of negative and positive weights.

In summary, the main contributions of this work are as follows:

i. We propose a novel SC accelerator for CNNs, called SkippyNN, which employs SC-based operations to significantly reduce the area and power consumption compared to binary implementations while preserving the quality of the results.
ii. To reduce the computation time and energy consumption (the two main challenges in efficient SC-based design), we introduce DMAC which uses the differences between successive weights to improve the speed of computations. Employing DMAC further omits the overhead cost of handling negative weights in the stochastic arithmetic units.
iii. Evaluating the performance of SkippyNN on four modern CNNs shows on average 1.2x speedup and 2.7x energy saving compared to the conventional binary implementation.

2 RELATED WORK

Efficient hardware accelerators for CNNs has become a highly debated topic recently [1, 5, 6, 8, 10]. Most of the recent proposed hardware use low-bitwidth arithmetic units in their datapath as CNNs are inherently tolerant of bit-width variations [5, 21, 24]. This eliminates the need for costly full-precision arithmetic units. Eyeriss [6] proposed a dataflow to minimize the power consumption of data accesses needed in the computations by exploiting the data reuse pattern in the inputs and weights of a layer. Stripes [10] introduced a bit-serial inner-product engine that dynamically tunes the precision of computations to maximize energy saving and performance at the cost of a slight loss in the network accuracy. To reduce the computation load in the activation units (each convolution layer is usually followed by an activation layer), SnaPEA [1] proposed a heuristic approach for early prediction of the activation units output. Plenty of works have been done on the sparsity in convolutional layers to reduce the power consumption by eliminating unnecessary multiplications where at least one of the operands is zero [18, 25].

Although the recently proposed architectures strove to reduce power consumption with minimum degradation in performance, utilizing them in embedded systems is still limited due to tight energy constraints and insufficient processing resources. SC is an appealing alternative design method to conventional binary design that not only meets the energy constraints of embedded devices but also is implemented via ultra low-cost hardware resources [2].

There have been some recent efforts in implementing SC accelerators for CNNs [4, 14, 19]. Authors in [23] introduced a new SC multiplication algorithm, called BISC-MVM, for matrix-vector multiplication. In [14], a fully parallel and scalable architecture for CNNs is proposed. The impact of low-bitwidth operations on the accuracy of SC-based CNNs is investigated in [22]. They develop a dynamic precision scaling method that achieves significant improvements over conventional binary implementations.

This work introduces SkippyNN, a SC accelerator for CNNs that effectively reduces the computation time of convolution by faster multiplication of bit-streams through *skipping* the unnecessary bit-wise ANDs. The time saving due to using the proposed *bit skipping* approach further improves the energy consumption (i.e., *power* × *time*) compared to the state-of-the-art work.

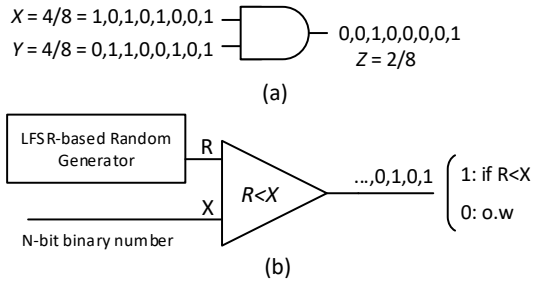

Figure 1: (a) Stochastic multiplication using AND gate. (b) LFSR-based SNG.

3 MOTIVATION AND BACKGROUND

3.1 Stochastic numbers

In the unipolar format, stochastic numbers (SNs) are interpreted as probabilities in the [0, 1] interval. In convolution computation, the inputs are the values of a feature map (commonly integers in the range [0, 255]). So a pre-processing step is required to scale the numbers to the [0, 1] interval. This is done by dividing the input numbers by 256. For instance, the input number $x = 23$ in the conventional binary domain is replaced by the number $x_s = 23/256$ in the stochastic domain. The values of the weight vectors are typically in the range [-1,1]. So there is no need to scale the weights when multiplying them by the inputs.

SNs are represented by streams of random (or unary [17]) bits where the ratio of the number of ones to the length of the bit-stream determines the value in the [0, 1] interval. Multiplication, as an essential operation in CNNs, is performed by bit-wise ANDing of SNs (bit-streams). This results in a significant reduction in the hardware cost compared to the conventional binary multiplier. A simple stochastic multiplication using a two-input AND gate is exemplified in Figure 1(a). Provided that X and Y are statistically independent (uncorrelated), a single AND gate can precisely compute $X \cdot Y$ [3].

3.2 Stochastic Number Generators

Converting binary numbers (BNs) to SNs and vice-versa are the primary steps in SC operations. As shown in Figure 1(b), a BN-to-SN converter (a.k.a Stochastic Number Generator or SNG) is often composed of a binary comparator and a linear feedback shift register (LFSR) as the random number generator (RNG) [3]. Employing different LFSRs (i.e., different feedback functions and different seeds) in generating SNs leads to producing sufficiently random and uncorrelated SNs. To convert an SN to BN, it suffices to count the number of 1s in the bit-stream. Therefore, a binary counter is a straight-forward circuit for SN to BN conversion.

3.3 State-of-The-Art Stochastic Multiplication

Stochastic multiplication of random bit-streams often takes a very long processing time (proportional to the length of bit-streams) to produce acceptable results. To mitigate this, authors in [23] proposed a new method, called *BISC-MVM*, that significantly reduces the number of clock cycles taken in the stochastic multiplication by using a novel SNG. In contrast to the conventional SNGs that use RNGs (e.g., LFSRs), the SNG in BISC-MVM follows a deterministic bit shuffling pattern and so, is isolated from random fluctuations. The proposed SNG is equipped with a finite state machine (FSM) and a multiplexer

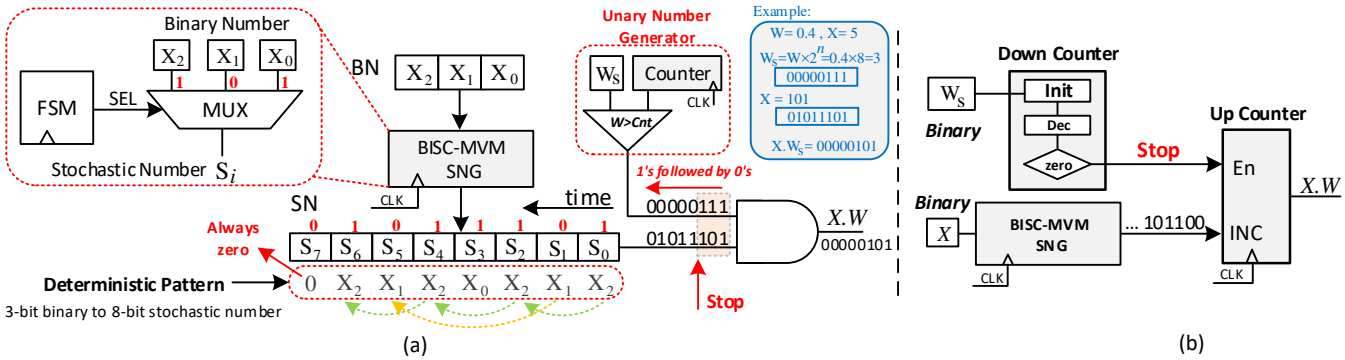

Figure 2: (a) Stochastic multiplier and SNG and (b) the MAC unit with BN-to-SN converter in BICS-MVM [23].

(MUX). Figure 2(a) illustrates the pattern generated by the SNG of BISC-MVM. We clarify the BISC-MVM method by an example.

Suppose that an n-bit BN, $x = x_{n-1}x_{n-2}...x_0$, is to be converted to a bit-stream S with a length of 2^n and multiplied by a fractional number w in the range [0,1]. Firstly, w is scaled by the scaling factor 2^n to achieve $w_s = 2^n \cdot w$. Then, w_s is transformed to a sequence of 1s followed by a sequence of 0s (so-called unary bit-stream). As shown in Figure 2(a), this pattern is simply produced by an unary number generator which is composed of a binary counter and a comparator. w_s acts as a mask vector for bit-stream S. x_{n-i} or the $(n-i)^{th}$ bit of x in the binary representation first appears in S at cycle 2^{i-1}, and thereafter in every 2^i cycles. The last bit of the bit-stream is always zero to guarantee that the value of the generated bit-stream is less than one [23].

The bit-stream produced at the output of the AND gate is the result of multiplying x by w_s. To calculate the value of $x \cdot w_s$, it suffices to compute the partial sum, P_{w_s}, by summing the first w_s bits of bit-stream S:

$$x \cdot w_s = P_{w_s} = \sum_{i=0}^{i=w_s} S_i \tag{1}$$

This approach is equivalent to what is presented in Figure 2(b). The multiplication can be simply performed by counting the number of 1s in the first w_s bits of S using an up counter. A down counter (which is initially set to w_s) is also needed to terminate the multiplication. Employing this algorithmic property leads to a significant reduction in the multiplication time since there is no need to wait to generate the entire bit-stream.

4 OUR PROPOSED ARCHITECTURE

A common CNN is composed of a large number of layers where convolutional layers constitute the most portion of the computation load and hardware cost. Figure 3(a) demonstrates a 2D convolution in which a set of 2D input feature maps (each of which is called ifmap channel [6]) is convolved with a set of 2D filters. Each point in the output feature map is achieved by sliding filters over the corresponding ifmap channels and calculating the inner-product of each window followed by a summation across all the channels. Due to a large number of multiplications in each layer, developing low-cost designs for these heavy operations is inevitable. Although employing BISC-MVM approach reduces the computation time of convolutions, further improvement in mitigating the computation load of multiplications is still needed. To this aim, our proposed architecture employs a novel differential Multiply-and-Accumulate unit, called DMAC, which further improves the processing time and energy consumption by using a differential multiplication algorithm.

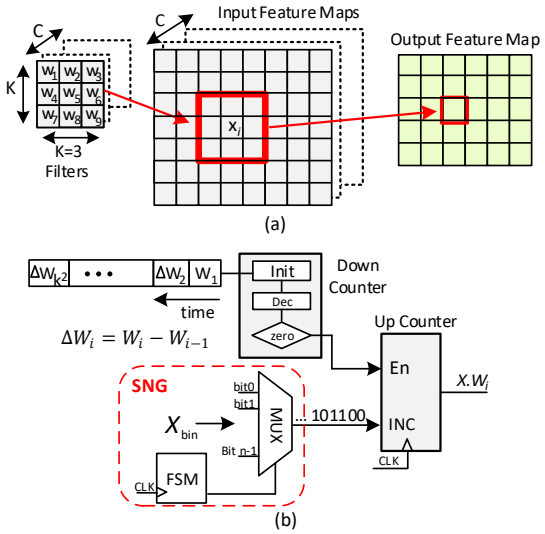

Figure 3: (a) A typical convolutional layer and (b) Our proposed DMAC

4.1 Differential MAC

As shown in Figure 3(a), a filter of size $C \times k \times k$ in a convolutional layer is composed of C channels, each channel is a 2D vector of size $k \times k$. Convolution is an inner-product where each input value x_i in the ifmaps is multiplied by the weights of the corresponding filter channel $(w_1, w_2, ..., w_{k\times k})$. So, each multiplication has an operand x_i in common with the other multiplications $(x_i \times w_2), ..., (x_i \times w_{k\times k})$. Using the architecture of BISC-MVM, the computation of $x_i \times w_j$ takes w_j clock cycles (x_i is fed to the SNG and the down counter is initially set to w_j). To multiply x_i by the successive weights of the filter, provided by maintaining the result of the first multiplication $(x_i \times w_1)$, we can calculate the next one simply as follows:

$$x_i \times w_2 = x_i \times w_1 + x_i \times (w_2 - w_1) \tag{2}$$

When $(w_2 - w_1)$ is less than w_2, the multiplication time reduces from w_2 to $(w_2 - w_1)$ clock cycles. Figure 3(b) illustrates the micro-architecture of the proposed DMAC. Note that the points (x_is) located in the border of the ifmaps are multiplied by a subset of the weights of the filter and this leads to a lower improvement compared to other points. However, the number of these points is far less than the number of inner points and therefore, their impact is negligible.

4.2 Reordering The Weights

Minimizing the differences between successive values in the weight vector leads to further reduction in the computation time of the

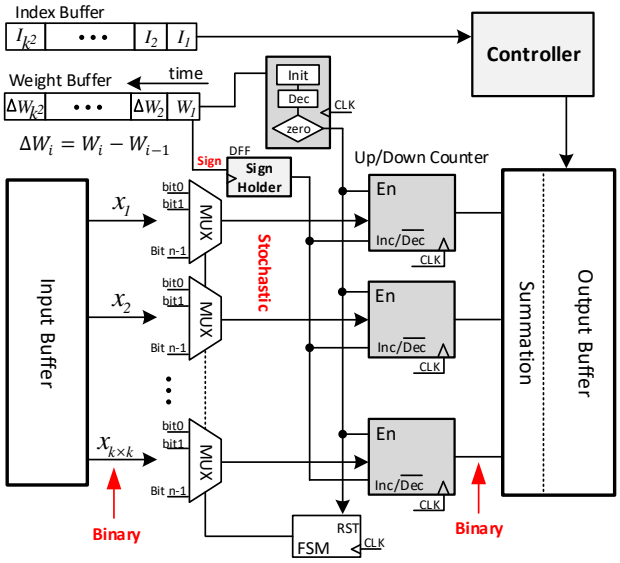

Figure 4: The micro-architecture of the SkippyNN

multiplications. To this end, we reorder the weights of a filter and fill the weight vector in ascending order. This reordering minimizes the differences between successive weights in the weight vector. When reordering the weights, an index buffer is also needed to hold the indices of the weights in the original filter.

4.3 SkippyNN Micro-architecture

Driven by the performance improvement of the proposed DMAC and the reordering technique, SkippyNN provides an accelerator that is capable of performing a 2D convolution efficiently. A high-level block diagram for the micro-architecture of SkippyNN is depicted in Figure 4. The *controller* is obligated to manage the timing and organize the entire system. SkippyNN is equipped with an *input buffer* to fetch the needed input data from the main memory, and an *output buffer* to store the corresponding results of each convolution. In what follows, each dedicated unit of the proposed accelerator is discussed in detail.

Index and Weight Buffers. To minimize the differences between successive weights, the weights must be sorted. This sorting is done offline and the sorted weights are loaded into the weight buffer. Since there is no priority among the multiplications in a convolution, this reordering does not have any impact on the output. However, the *controller* should be aware of the proper ordering using the *index buffer*. In every cycle, the *controller* fetches a weight and broadcasts it to all the counters. After completing the multiplications, the *controller* stores the result corresponding to each index.

BN-to-SN Converter (SNG). Since the ifmaps are entirely independent of each other, there is no need to guarantee that the generated SNs are uncorrelated. Given this insight, we share a single FSM among the SNGs of all the input operands (ifmaps) to control all the multiplexers. As shown in Figure 4, the process of generating the bit-streams is terminated when the down-counter reaches its zero state.

Sign Holder. Usually, the weights of the filters are bipolar and in the [-1,1] interval. Therefore, the accelerator architecture should support signed multiplication. To this end, we use up/down counters for multiplications to be able to either decrease or increase the output. In every cycle, if the weight is negative, the counter counts in descending order; otherwise, the counter counts in ascending order. Note that the inputs to the convolution layers are always positive. In the first layer, the input data is the pixels of an image and so greater than zero. Each convolutional layer in modern CNNs is followed by a ReLU (Rectified Linear Unit) activation function. The ReLU activation function returns zero for the negative inputs and returns the input itself (unchanged) when the input is greater than zero. Thus, the intermediate ifmap values (the inputs to the middle convolutional layers) are also positive.

When using the differences of successive weights (instead of the original weights), all the values inside the weight buffer are positive except the first one. After sorting the weights in ascending order, the first value of the weight buffer is the smallest weight which is typically a negative value. The remaining values are all positive ($w_2 > w_1 \Rightarrow \Delta w = w_2 - w_1 > 0$). To support the first bipolar multiplication, SkippyNN is equipped with a sign holder unit (a simple D-FlipFlop) that holds the sign of the first value. The sign holder is connected to the *Inc/Dec* input of the counters to determine if they should count up or down.

Counters and Summation Unit. The designed counters are used to multiply the ifmaps by the weights and convert the results to BNs simultaneously. The counters are followed by a summation unit which adds the results of the multiplications with respect to the original order of the weights. The final results are stored in the *output buffer* for the next layer.

5 EXPERIMENTAL RESULTS

To evaluate the performance of SkippyNN, we use four modern CNNs: AlexNet, VGG16, Inception-V3 and MobileNet. We first analyze the network accuracy when employing the proposed SC-based design. Then, we evaluate the synthesis results of hardware implementation of SkippyNN. Finally, we evaluate the speedup and energy-efficiency of SkippyNN compared to BISC-MVM and the conventional binary implementation.

5.1 Network Accuracy

Figure 5 shows the accuracy of SkippyNN versus BISC-MVM and conventional binary implementation for different bitwidths. We used Caffe [9] to measure the overall accuracy of the networks by processing 10,000 randomly selected images from ImageNet [20]. The training process is executed offline to extract the network parameters. The uncertainty of the SC-based multipliers caused no considerable accuracy loss due to the inherent tolerance of neural networks to inaccuracy in the computations.

Our experimental results show that the accuracy of SC-based implementations (both SkippyNN and BISC-MVM) are comparable to the conventional binary implementation. Comparing SkippyNN and BISC-MVM, the proposed SkippyNN showed a slightly lower accuracy. The reason is in processing shorter bit-streams in the SkippyNN design. Generally, in a SC-based multiplication, the accuracy is directly proportional to the length of the bit-stream. In BISC-MVM, the inputs (x_is) are multiplied by the original weights (i.e., w_i). In SkippyNN, they are multiplied by the differences of successive (and reordered) weights (i.e., $w_i - w_{i-1}$). Considering the fact that $w_i > w_i - w_{i-1}$, the bit-stream length of w_i is also greater than that of $w_i - w_{i-1}$. The shorter bit-stream length of the operands in SkippyNN may induce a bit of uncertainty in the results. However, as shown in Figure 5, due to the approximate tolerance of neural networks, this does not have a considerable impact on the network accuracy.

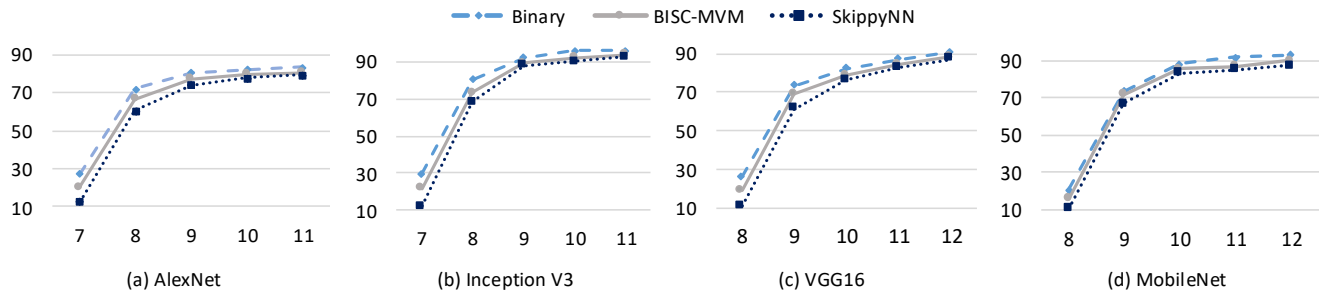

Figure 5: Comparison of the network accuracy (top-5) of the binary implementation versus BISC-MVM and our proposed SkippyNN. The horizontal axis shows the bitwidth and the vertical axis shows the percentage of the accuracy.

Table 1: The synthesis results in 45nm technology: area (μm^2), critical path latency (ns), power (mW) and energy/cycle (pJ) of a convolution engine (3×3 filter). The results are shown for different bitwidth of the operands (weights and ifmaps).

Arch.	8-bit				9-bit				10-bit				11-bit			
	Area (μm^2)	CP (ns)	Power (mW)	EPC (pJ)	Area (μm^2)	CP (ns)	Power (mW)	EPC (pJ)	Area (μm^2)	CP (ns)	Power (mW)	EPC (pJ)	Area (μm^2)	CP (ns)	Power (mW)	EPC (pJ)
BISC	1408	0.93	1.369	1.27	1585	0.93	1.530	1.42	1778	0.94	1.665	1.56	1938	0.94	1.828	1.71
Skippy	1439	0.93	1.393	1.29	1616	0.93	1.554	1.44	1809	0.94	1.689	1.58	1969	0.94	1.852	1.74
Binary	5429	1.47	3.287	4.83	6657	1.6	3.726	5.96	8370	1.73	4.301	7.44	9232	1.95	4.515	8.80

5.2 Hardware Cost Evaluation

To evaluate the hardware cost of SkippyNN, we developed a cycle-accurate micro-architectural simulator in RTL Verilog. Synthesis results were extracted by the Synopsys Design Compiler using a 45nm technology library and measured for different bitwidths in SkippyNN, BISC-MVM and conventional binary implementation. As reported in Table 1, due to their simpler arithmetic units, SC-based implementations (SkippyNN and BISC-MVM) offer a significant improvement in the area, working frequency (1/critical path latency) and power consumption. SkippyNN and BISC-MVM deliver, on average, 4.4x improvement in energy/cycle (the product of critical path latency and power consumption) compared to the binary implementation. As the bitwidth gets longer, the working frequency of the binary convolution engine decreases. The working frequency of SC-based implementations, however, remains relatively unchanged as the complexity of the stochastic design is independent of the precision of data.

Comparing SkippyNN and BISC-MVM in terms of area and power consumption, the proposed architecture has a slightly higher hardware cost due to using the *index buffer* which is responsible for holding the indices of weights. Considering the improvement in the processing time, this slightly higher hardware cost is negligible.

5.3 Performance and Energy Improvement

Although SC-based convolution engines benefit from low-cost hardware implementations, their high processing time and energy consumption are still the main challenges when employing them in hardware accelerators. Processing time is obtained by multiplying the number of clock cycles taken in a convolution and the critical path latency. Figure 6 shows the speedup of SkippyNN and BISC-MVM versus the binary implementation. SkippyNN uses the differences of the reordered weights and hence, takes fewer number of cycles than BISC-MVM. Comparing the processing time of SkippyNN and the binary design is however debatable. In SkippyNN, the number of clock cycles taken in a convolution engine is $w_1 + \Delta w_2 + \Delta w_3 + \ldots + \Delta w_{k^2}$. In the binary implementation, however, this number is k^2 (equivalent to the size of the filter, $k \times k$). Therefore, in convolutional layers with large filters, the processing time of binary implementation is relatively high. The bitwidth also plays a key role in the processing time of SkippyNN. As the bitwidth gets shorter, the weights get closer to each other (since they are being scaled down to a lower range). Hence, the differences of successive weights and consequently the required number of cycles decrease. As can be seen in Figure 6, due to these reasons, in some cases the SkippyNN has shown a lower processing time than the conventional binary implementation.

Table 2: Performance evaluation of SkippyNN. The numbers are normalized to the binary implementation.

CNN	AlexNet	Inception-V3	VGG16	MobileNet
Bitwidth	8-bit	8-bit	9-bit	10-bit
Speedup	1.24x	0.82x	1.54x	0.05x
Energy Reduction	2.92x	1.47x	3.7x	0.12x

Energy consumption is evaluated as the product of the number of clock cycles and the energy per cycle. Similar to the processing time, energy reduction from the proposed architecture depends on the filter size and bitwidth. As illustrated in Figure 7, in some CNNs, SkippyNN consumes a lower energy than the binary implementation. For instance, in AlexNet, when using 8-bit precision operations, SkippyNN offers over 1.2× energy reduction compared to the binary design while providing acceptable network accuracy.

In MobileNet, the improvement in terms of energy and processing time degrades considerably. The filter size is the main reason behind this observation. MobileNet consists of several convolutional layers with filters of size 1×1. Therefore, using the differences of the weights is impractical and SkippyNN is unable to reduce the number of processing cycles. Table 2 summarizes the overall speedup and energy reduction of SkippyNN compared to the conventional binary implementation. For each CNN, a bitwidth is selected that maximizes the speedup and energy reduction while providing comparable network accuracy (except MobileNet as a negative point).

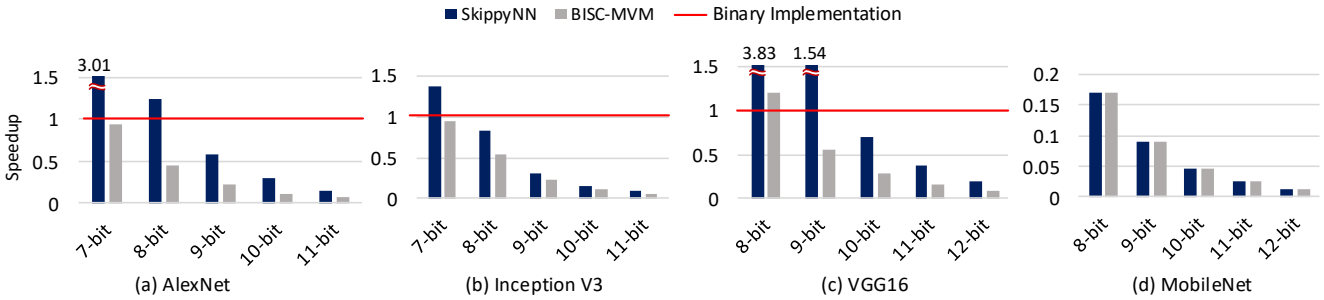

Figure 6: Overall speedup of SkippyNN and BISC-MVM. The results are normalized to the binary implementation.

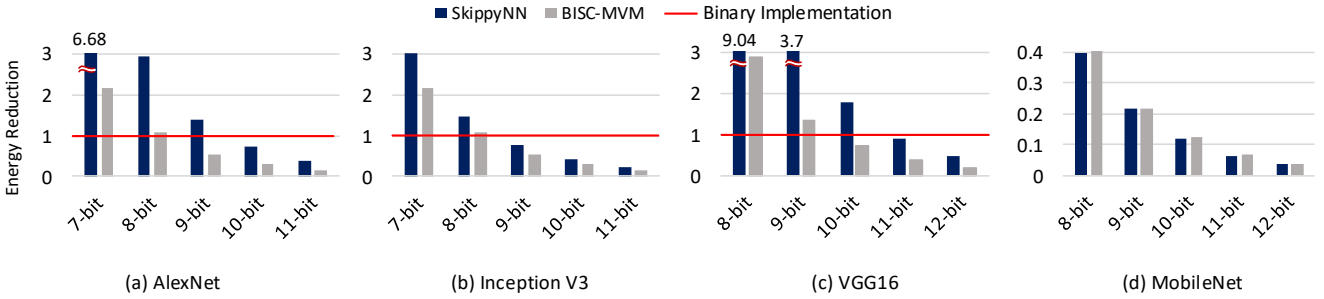

Figure 7: Overall energy reduction of SkippyNN and BISC-MVM. The results are normalized to the binary implementation.

6 CONCLUSION

In this work, we proposed a novel SC accelerator for hardware implementation of CNNs. SkippyNN aims at reducing the computation time of the multiplications in the convolutional layers of CNNs. We showed that the input data can be multiplied by the differences of successive weights instead of the original weights. Exploiting this arithmetic property, we achieved a significant reduction in the processing time and energy consumption. SkippyNN outperforms the conventional binary implementation by delivering, on average, 1.2× speedup and 2.7× energy reduction (except MobileNet as a negative point). Experimental results showed that the proposed architecture has no considerable impact on the neural network accuracy.

REFERENCES

[1] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, H. Esmaeilzadeh, and R. Gupta. 2018. Snapea: Predictive early activation for reducing computation in deep convolutional neural networks. ISCA.

[2] A. Alaghi and J. P Hayes. 2013. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)* 12, 2s (2013), 92.

[3] A. Alaghi, W. Qian, and J. P Hayes. 2018. The promise and challenge of stochastic computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 8 (2018), 1515–1531.

[4] M. Alawad and M. Lin. 2016. Stochastic-based deep convolutional networks with reconfigurable logic fabric. *IEEE Transactions on multi-scale computing systems* 4 (2016), 242–256.

[5] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Notices* 49, 4 (2014), 269–284.

[6] Y. Chen, T. Krishna, J. Emer, and V. Sze. 2016. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference, ISSCC 2016*. 262–263.

[7] S R. Faraji, M H. Najafi, B. Li, K. Bazargan, and D. J Lilja. 2019. Energy-Efficient Convolutional Neural Networks with Deterministic Bit-Stream Processing. In *Design, Automation, and Test in Europe (DATE)*.

[8] R. Hojabr, M. Modarressi, M. Daneshtalab, A. Yasoubi, and A. Khonsari. 2017. Customizing Clos Network-on-Chip for Neural Networks. *IEEE Trans. Comput.* 66, 11 (2017), 1865–1877.

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

[10] P. Judd, J. Albericio, T. Hetherington, T. M Aamodt, and A. Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[11] V. T Lee, A. Alaghi, J. P Hayes, V. Sathe, and L. Ceze. 2017. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 13–18.

[12] B. Li, M. H. Najafi, and D. J. Lilja. 2016. Using Stochastic Computing to Reduce the Hardware Requirements for a Restricted Boltzmann Machine Classifier. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16)*. ACM, New York, NY, USA, 36–41.

[13] B. Li, M. H. Najafi, and D. J. Lilja. 2019. Low-Cost Stochastic Hybrid Multiplier for Quantized Neural Networks. *J. Emerg. Technol. Comput. Syst.* 15, 2, Article 18 (March 2019), 18:1–18:19 pages.

[14] Ji Li, Ao Ren, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, and Yanzhi Wang. 2017. Towards acceleration of deep convolutional neural networks using stochastic computing.. In *ASP-DAC*. 115–120.

[15] P. Li, D. J Lilja, W. Qian, K. Bazargan, and M. D Riedel. 2014. Computation on Stochastic Bit Streams Digital Image Processing Case Studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 3 (2014), 449–462.

[16] Y. Liu, Y. Wang, F. Lombardi, and J. Han. 2018. An energy-efficient stochastic computational deep belief network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 1175–1178.

[17] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan. 2018. Low-Cost Sorting Network Circuits Using Unary Processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 8 (Aug 2018), 1471–1480.

[18] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W Keckler, and W. J Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 27–40.

[19] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan. 2017. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. *ACM SIGOPS Operating Systems Review* 51, 2 (2017).

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[21] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh. 2018. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE.

[22] H. Sim, S. Kenzhegulov, and J. Lee. 2018. DPS: dynamic precision scaling for stochastic computing-based deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, 13.

[23] H. Sim and J. Lee. 2017. A new stochastic computing multiplier with application to deep convolutional neural networks. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*. IEEE, 1–6.

[24] A. Yasoubi, R. Hojabr, and M. Modarressi. 2017. Power-efficient accelerator design for neural networks using computation reuse. *IEEE Computer Architecture Letters* 16, 1 (2017), 72–75.

[25] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.

[26] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).