Scalable FPGA Accelerator for Deep Convolutional Neural Networks with Stochastic Streaming

Mohammed Alawad, *Member, IEEE,* and Mingjie Lin, *Member, IEEE*

Abstract—FPGA-based heterogeneous computing platform, due to its extreme logic reconfigurability, emerges to be a strong contender as computing fabric in modern AI. As a result, various FPGA-based accelerators for deep CNN—the key driver of modern AI—have been proposed due to their advantages of high performance, reconfigurability, and fast development round, etc. In general, the consensus among researchers is that, although FPGA-based accelerator can achieve much higher energy efficiency, its raw computing performance lags behind when compared with GPUs with similar logic density. In this paper, we develop an alternative methodology to efficiently implement CNNs with FPGAs that outperform GPUs in terms of both power consumption and performance. Our key idea is to design a scalable hardware architecture and circuit design for large-scale CNNs that leverages a stochastic-based computing principle. Specifically, there are three major performance advantages. First, all key components of our deep learning CNN are designed and implemented to compute stochastically, thus achieving excellent computing performance and energy efficiency. Second, because our proposed CNN architecture enables a stream-mode computing, all of its stages can process even the partial results from preceding stages, therefore not incurring unnecessary latency due to data dependency. Finally, our FPGA-based deep CNN also provides a superior hardware scalability when compared with conventional FPGA implementations by reducing the bandwidth requirement between layers.

The results show that our proposed CNN architecture significantly outperforms all previous FPGA-based deep CNN implementation approaches. It achieves 1.58x more GOPS, 6.42x more GOPS/Slice and 10.92x more GOPS/W when compared with state-of-the-art CNN architecture. The top-5 accuracy of stochastic VGG-16 CNN is 86.77% with 18.91 fps frame rate.

Index Terms—Convolutional Neural Network, FPGA, Stochastic Computing

1 Introduction

Large-scale deep learning (DL) has recently attracted tremendous amount of research interests largely due to its unprecedented success in almost all computer vision tasks, such as image classification [1] and face recognition [2], and natural language processing (NLP) tasks, such as sentence classification [3] and semantic parsing [4]. Among many deep learning methodologies, convolutional neural networks (CNNs), due to their significant improvement in accuracy relative to other more traditional machine learning algorithms, have been extensively studied by both the academic community [1], [5] and industry like Google [6], Microsoft [7] and Facebook [8].

Conceptually, CNNs are hierarchically structured with multiple feature extraction and classification stages. Furthermore, each feature extraction consists of multiple convolution layers arranged in sequence that are interspersed with non-linear activation and subsampling layers, while the classification stage consists of one or several fully connected neural network layers at the end. Not surprisingly, the impressive computing power of a state-of-the-art deep network lies in its huge number of floating-point operations performed, its enormous connectivity between convolution layers, and its huge storage requirements. All these factors, coupled with high energy efficiency needed, severely limit large-scale CNN's applicability in many real-time embedded applications. For example, AlexNet [1], which is used for ImageNet dataset, consists of almost 650,000 neurons and 60 million synapses, and involved around 2-4 GOPS per image classification. Consequently, hardware accelerators using GPUs, FPGAs, ASICs, and other specialized computing fabrics, become invaluable in many energy-constrained and performance-centric applications. In fact, many studies with these non-CPU methods have reported order-of-magnitudes improvements in energy efficiency and computing performance over conventional general-purpose CPUs. In particular, GPUs are widely used because they are very efficient in realizing floating-point matrix multiplication operations which are the core computing element of deep CNNs; however, they are inefficient in terms of energy consumption [9], especially for current networks, where matrix multiplications for convolution process consume more than 90% of the overall computation time [10]. Therefore, discovering more energy-efficient hardware means to implement large-scale CNNs without compromising computing performance remains a critically important research challenge.

Among various hardware implementation mediums, Field Programmable Gate Array (FPGA) fabrics, which com-

- *M. Alawad is with the Health Data Sciences Institute, Oak Ridge National Laboratory, Oak Ridge, TN, 37831.*
  E-mail: alawadmm@ornl.gov
- *M. Lin is with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, 32816.*
  E-mail: mingjie@eecs.ucf.edu

This manuscript has been authored in part by UT - Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid- up, irrevocable, world-wide license to publish or reproduce the published form of the manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public- access-plan).
Manuscript received April 20, 2018; revised October 17, 2018.

bine many benefits of both VLSI and general-purpose soft-processor, have been identified as very attractive platforms for accelerating CNNs computing. The main advantages of using FPGA platforms can be summarized as follows: 1) FPGAs provided superior energy efficiency than GPUs for Deep CNNs [9]. 2) FPGA technologies are advanced significantly by increasing the hardware processing units, on-chip memories, and higher memory bandwidth which reduce the performance gap between FPGA and GPU platforms. 3) FPGA architectures are flexible and can handle irregular parallelism for emerging deep CNNs and custom-defined data types that are difficult to be handled by GPUs [11]. 4) Current FPGA design tools support high level programming instead of RTL, which makes FPGAs accessible to developers [12].

Unfortunately, although the conventional FPGA-based implementation of CNNs can achieve outstanding energy efficiency, its computing performance is typically not as good as GPU-based CNNs. In fact, there are two major limitations of using FPGAs to realize CNNs conventionally. First, FPGAs have limited availability of DSP slices that can be used for multiply and accumulate purposes. In fact, orders of magnitude of the available DSP slices are required to realize efficient CNNs and achieve high level of parallelism. For example, if the input and output feature maps of a layer are P and Q respectively, then $P \times Q$ convolvers are required to achieve the parallelism in CNNs, which requires a tremendous hardware resources. Therefore, they need to be shared among several neurons [13]. Unfortunately, this can significantly reduce the performance of the circuit, where limited number of parallel neurons can be realized on-chip, and loop tiling technique is adopted to reuse the hardware resources for computation and store all intermediate data off-chip in external memory chips. Limited hardware resources can also lead to an inefficient design since same feature maps and/or filter kernels need to be read from off-chip memory so many times. Second, since the intermediate results are used in the following layer, a huge memory is required to save this data. This also leads to a high memory communication rate which requires more energy consumption, where off-chip memory access consumes orders of magnitude more energy than on-chip memory access [14], [15]. Even if there are enough hardware resources to realize the whole computing units in CNNs by connecting multi FPGAs, the connection between them will be very complex and unrealizable which limits the scalability of the design and does not ensure performance improvement when increasing the available hardware resources. Therefore, different studies have been presented recently to overcome this issue.

Given all these performance challenges, a new hardware implementation strategy is needed. Specifically, because the computing efficiency of matrix multiplications on GPU platforms is almost unparalleled, an alternative way for computing is required to implement CNNs with FPGA platforms instead of employing FPGAs to imitate the functionality of GPUs. The other reason why we need to think differently is the bandwidth limitation of FPGAs (about 10 to 20 GBPS [16]) compared to GPUs(up to 700 GBPS [17]). According to [18], using matrix multiplication to do convolution results 25 times more data volume for the input feature maps in the CONV layer, which reduces the advantages of FPGA accelerators. In this paper, we exploit the underlying performance benefits of stochastic-based computing principles with FPGA fabric that transcend beyond deterministic CMOS-based computing [19] and possess many notable advantages. Specifically, stochastic-based computing units typically use simple circuit units to conduct complex operations, which leads to area and energy-efficient designs suitable for parallel processing. Moreover, stochastic-based computing units are able to tolerate device failures and construct robust circuits against large device variations. Despite of many apparent benefits, stochastic-based computation paradigms suffer from accuracy degradation, where the results accuracy of stochastic-based computing is proportional to the length of stochastic stream. Fortunately, stochastic-based computing can be utilized in applications that can tolerate a certain degree of error, such as machine learning algorithms, signal processing and control applications and we will show in Section 3 how it can be used to realize CNNs.

Our motivation is to propose an alternative network architecture and computing algorithm for realizing efficient accelerator for deep CNNs to avoid the limitations of conventional computing methods. In this paper, we propose a synthesized stochastic-based design for deep CNNs, the state-of-the-art machine learning algorithm, by applying the stochastic-based computing principles on all computing components. To do so, all the layers of a CNN need to be converted to the stochastic domain. Thus, the complicated structure of conventional CNNs is replaced with a channel of simple stochastic processing units that process streams of random samples [20]. Besides the advantages that can be achieved by using stochastic-based computation, such as high energy efficiency, low area cost, simplicity and robustness, our proposed SCNN resolves the computation and communication bounds in conventional deep CNN hardware realizations. Specifically, avoiding the loop tiling required to fit a small portion of data on-chip, eliminating the need for storing the huge intermediate data results off-chip, reducing the number of kernels that need to available on-chip, reducing the connections complexity between layers, and significantly reduce the hardware resources required for each convolution circuit. The other advantage of our proposed architecture, as compared to prior FPGA acceleration studies on CNN, is that it accelerates the three main layers of CNN models, convolution, activation and pooling. This eliminates the diminish of overall energy-efficiency and performance gains result from accelerating the convolution layer only [18]

The rest of the paper is organized as follows. Section 2 presents the related works. In Section 3, the background theory of CNN architectures and stochastic CNNs is presented. Section 4 describes our proposed stochastic-based CNN architecture, and Section 5 presents the error analysis. Results and analysis are discussed in Section 6. Then, we conclude in Section 7.

2 Related Works

CNNs recently draws a lot of attention due to their great success in object recognition, semantic segmentation, and

obstacle detection especially because of the availability of large training dataset and powerful computing platforms. Therefore, several research works have been presented recently to realize efficient CNNs on various platforms, including GPU [21], FPGA [22], and ASIC [23]. Numerous studies have found that one main restriction of a CNN's performance is its convolution step [24]. As such, a large number of research works have been presented to improve the hardware implementation of CNNs and reduce the computational cost. Systolic structure has been used to realize the convolution operation in CNNs [25], [26], however it requires high memory bandwidth and does not support the flexibility of CNNs' settings [23], [27]. Interconnection with memory has also been considered in [28] as another limitation in realizing CNNs. They observed that it is the key for realizing efficient hardware accelerators in terms of performance and energy consumption. Single Instruction Multiple Data (SIMD) architecture has been used to perform the convolution operation to minimize the required memory bandwidth. In [27], they propose a configurable accelerator template for CNNS that can optimize the on-chip memory size and data reuse. In [28], loop tiling has been used with on-chip SRAM buffers to reduce off-chip memory communications. However, frequent memory access is required because the feature maps are treated as 1D data. In [23], researchers have eliminated all DRAM access for weight by storing all the weights in on-chip memory. Their architecture was tested on simple task CNNs only. Therefore, it fails when considering state-of-the-art CNNs for large scale visual recognition, which has millions of weights. Although Zhang et al. [22] have tried to improve both the computation engine and communication issue, their CNN implementation is based on deterministic computing by using loop tiling and data reuse, therefore their performance are severely limited by hardware budget and memory bandwidth. In summary, most researches have focused on improving the performance of either computing circuit or the global communication with the off-chip memory issue. Although a few of them have tried to improve both, such as [22], these studies all follow the conventional way to realize CNNs and suffer from the high computational complexity, memory bandwidth, off-chip memory access, and inter-layers connections complexity challenges.

The restrictions in conventional implementation of deep CNNs using FPGAs lead to the need to develop the next generation architectures for deep learning algorithms. Compact data types have been proposed to represent data using 4-8 bits instead of using 32-bit single precision floating point with a small reduction in accuracy [29], [30]. Lower number of bits is used in ternary neural networks [31]. They use 2 bits to represent weights, however neuron values are represented by 32 bits. Researchers have also exploited network sparsity to improve the efficiency. Data sparsity results from the presence of zeros in neuron values and weights [32]. Mostly ReLU units produce zeros from negative neuron values. This approach requires less operations due to the use of sparse matrix multiplication instead of dense ones. Another approach to exploit sparsity is to prune the network weights that are considered not important by making them zeros [33]. These schemes have irregular parallelism and custom data type properties which make them difficult to be implemented using GPUs, but suitable for FPGA realization [9]. They all can be considered when designing our proposed scheme to achieve more efficiency, however in this paper, we focus on the conventional CNN architecture.

Very recently, stochastic-based computing has been presented to realize CNNs [34]. Their proposed structure can significantly improve the performance and energy consumption of simple task CNNs [35], [36]. However, it follows the conventional structure of realizing CNNs when considering state-of-the-art very deep CNNs, such as AlexNet [1] and VGG modules [37]. Therefore, it could not solve the scalability and interconnection problems. In this paper, we take the advantage of stochastic-based computing to reduce the hardware computing resources and connections complexity, and eliminate the intermediate data storing requirement and propose a scalable CNN architecture to realize very deep CNNs.

3 Background

3.1 Convolutional Neural Networks

A practical convolutional neural network typically consists of several feature extraction layers, each of which normally contains several convolution and nonlinear activation layers, and one optional pooling or sub-sampling layer. Fig. 1 illustrates the composition of one feature extraction stage. Specifically, **convolution layer** extracts features from all local regions of input image by convolving a 2-D filter kernel over the input image. The filter kernels are obtained from the training phase, which are assumed to be known in this paper. We only focus on the feed-forward computation in a CNN. More specifically, we construct each convolution layer with parallel convolutional neurons, each of which can be represented as 3D filtering with different kernel coefficients on the input feature maps. Subsequently, this step produces different feature maps for the next layer, the first dimension is the number of input feature maps and the other two are the height and width of the feature maps. Right next to a convolution layer, a **nonlinear activation layer** transforms a resulting feature map through a predefined nonlinear function. This is to mirror the biological mechanism of subduing and enhancing neural signals within a biological brain. Finally, an optional **pooling layer** usually follows in order to reduce the resolution of the convolved image. The pooling layer takes small rectangular blocks from the convolution layer and subsamples it to produce a single output from that block.

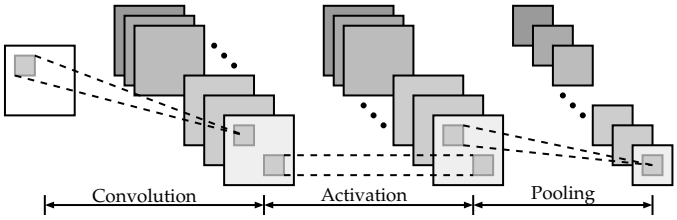

Fig. 1: Architecture diagram of convolutional neural network with three standard processing layers: convolution, activation, and pooling.

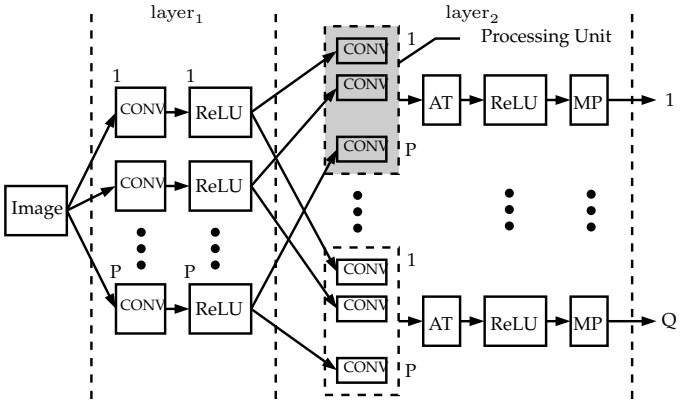

Fig. 2: Conventional structure of the first two layers of ConvNet, where CONV is convolution circuit, AT is adder tree, ReLU is rectified linear unit and MP is max pooling.

3.2 Complexity of CNNs

There are three types of parallelism in conventional CNNs [24] need to be achieved to obtain efficient realization. The parallelism in convolution operations, the parallelism in combining many convolved input feature maps and producing one output feature map, and the parallelism in computing many outputs independently. To our knowledge, there is no efficient hardware architecture that can achieve all these levels of parallelism because of the limitations in hardware resources and memory bandwidth if computing in a deterministic domain. We will consider the structure shown in Fig. 2 to demonstrate the limitations of achieving the three parallelism types in conventional hardware-based realization of CNNs. If a layer has P and Q input and output feature maps respectively, then $P \times Q$ connections are required to connect it with the previous layer, where P and Q are big numbers. To process all the inputs simultaneously, P convolvers are required, each one needs k^2 multipliers and $k^2 - 1$ adders to achieve the parallelism in convolution operation. The outputs of the convolvers are combined by the adder tree circuit, which also requires $P - 1$ adders, to produce one output feature map. Therefore, the hardware complexity of each processing unit in conventional CNNs is Pk^2 multipliers and $P(k^2 - 1) + P - 1$ adders. To achieve the parallelism in computing many outputs independently, Q processing units are required. On the other hand, this layer requires (k^2PQ) memory words to store the kernel coefficients. We can see from that the hardware resources requirement to achieve all the types of parallelism is tremendous. Even if we have enough hardware resources by connecting many FPGAs, the connections between layers is another obstacle that limits the scalability. Current FPGA-based CNNs architectures, such as [22], [30], realize some processing units on-chip and reuse them for different tasks. These architectures produce a huge intermediate results that need to be stored off-chip which degrades the efficiency of CNNs, and also they need to read the same feature map so many times. Therefore, we propose an alternative computing methodology to avoid the limitations of realizing CNNs conventionally.

3.3 STOCHASTIC-BASED CNNs

In this section, we present the methodology to stochastically compute all three layers of a feature extraction stage. In the convolution layer, we exploit the well-known convolution theorem in random processes to significantly reduce its computing complexity. Furthermore, in both nonlinear activation layer and pooling layer, we again only perform simple probabilistic operations to either pass or eliminate random samples. Note that, in all these three layers, only random samples are being processed. Therefore, the whole computing procedure exhibits a streaming mode. In other words, each layer doesn't have to wait for the completion of its preceding layer in order to proceed, therefore completely being pipelined. This contrasts sharply with the deterministic version of these layers.

Stochastic-based CNN methodology has several advantages. First, because all information are encoded probabilistically, many operations are further decoupled and its overall performance can be significantly improved. Secondly, the decoupling due to stochastic processing can improve the overall scalability of our CNN-based deep learning, which potentially can greatly reduce its total memory footprint. Finally, throughout our CNN-based deep learning system, no complicated operations such as multiplication or division are required. Instead, All operations are integer-based. The real value of the input data is encoded by probability values determined by the proportion of a particular random samples.

3.3.1 Stochastic-Based Convolution Layer

Stochastic-based multi-dimensional convolution leverages the probabilistic principle that the probability density function (PDF) of the sum of two or more independent random variables is the convolution of their individual PDFs [38]. These PDFs may have the same size or each one has a different size. Therefore, we can use this theorem to convolve kernels with feature maps stochastically in CNNs. We define an n-dimensional random variable by mapping the outcomes of the space to an n-dimensional space, thus obtaining the n-dimensional random variable $\mathbf{X} = [X_1, X_2, \cdots, X_n]$, where X_1, X_2, and X_n are 1-D random variables.

The above theorem proves that, through interpreting input waveforms as probability density functions, a conventional multi-dimensional convolution in spatial-temporal domain can be readily translated into a number of independent parallel additions in probabilistic domain. To further discuss this method, suppose $\mathbf{X}$ and $\mathbf{Y}$ are two n-dimension vectors treated as two n-dimensional probability density functions $m_{\mathbf{X}}$ and $m_{\mathbf{Y}}$, respectively. For each of these, we generate large ensembles of random samples accordingly, $T_{X_1}, T_{X_2}, \cdots, T_{X_n}$ and $T_{Y_1}, T_{Y_2}, \cdots, T_{Y_n}$, we will discuss this process in details in Section 4. Finally, corresponding to each dimension $1, 2, \cdots, n$, we add random samples T_{X_k} and T_{Y_k} to generate a new set of random samples T_{Z_k}, where $k = 1, 2, \cdots, n$. Subsequently, we extract the n-dimensional PDF of $T_{\mathbf{Z}}$, $m_{\mathbf{Z}} = \mathbf{X} * \mathbf{Y}$. In this paper, we considered the stride size equals to one, which is the most compute-intensive case. However, the proposed approach can support larger strides too. The cost will be having redundant computation. The required circuit to exclude

redundant numbers is very simple and cheap in terms of hardware resources and computation complexity. Also, since the resultant random samples from one convolution layer are passed to the next layer, indices range should be changed by considering the stride size.

3.3.2 Stochastic-Based Nonlinear Activation Layer

The nonlinear activation layer within a CNN is designed to mimic the biological mechanism of subduing and enhancing neural signals within a brain. Since its introduction, various activation transfer functions have been proposed and investigated, although which function is the best remains unsolved and most likely be application-specific. In this paper, we refrain from discussing which transfer function is the optimal choice because it is highly dependent on the specific input data set and the particular CNN topology. Instead, we present a detailed multi-phase pumping circuit to implement a widely-adopted linear rectifier activation function and demonstrate its effectiveness and hardware efficiency.

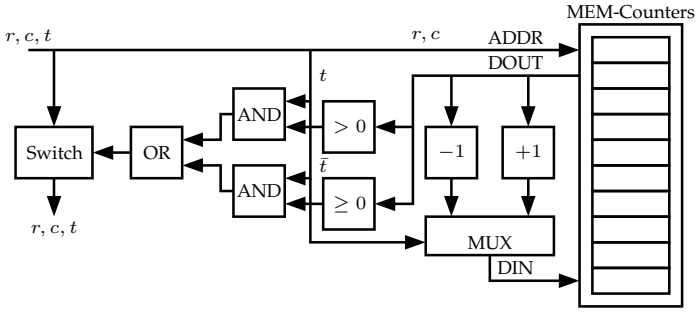

Fig. 3: Stochastic-based computing circuit for linear rectification (ReLU), where r, c, t represent the random row-column indexes and the corresponding sign bit, respectively.

Nair and Hinton [39] refer to neurons with linear rectifier nonlinearity as Rectified Linear Units (ReLUs). Various research studies have shown that, with ReLUs, a four-layer convolutional neural network can significantly reduce its overall training time, or the number of iterations required to reach 25% training error on the CIFAR-10 dataset, by about 4 times when compared with its equivalent network that uses tanh functions. In this study, the fact that all neural signals are encoded probabilistically allows us to perform such a linear rectification completely within the stochastic domain by directly manipulating the random samples passing through the CNN. Specifically, we maintain an accumulative counter $\texttt{CT}[r][c]$ in each neuron i. For each incoming random sample $s(r, c, t)$, depending on its sign $\texttt{sign}(s) = t$, we decide its passing or blocking accordingly by taking into consideration the current state of $\texttt{CT}[r][c]$. Fig. 3 shows the circuit diagram of stochastic-based ReLU. This algorithm has been thoroughly verified with both simulation and hardware implementation.

3.3.3 Stochastic-Based Pooling Layer

Pooling layers in CNNs are designed to summarize the neuron outputs that belong to neighboring groups in the same kernel map. More specifically, one can think that a pooling layer consists of a grid of pooling units that are spaced t_x pixels apart. Therefore, each neuron result after pooling summarizes a block of size $k \times k$ pixels centered at the location of the pooling unit. There are several ways to perform pooling methods, max and average are the most frequently used pooling methods. In max pooling method, its output is given by the maximum activation over non-overlapping rectangular regions of size (K_x, K_y), while in average pooling method, the output is computed by averaging the values in the region of (K_x, K_y).

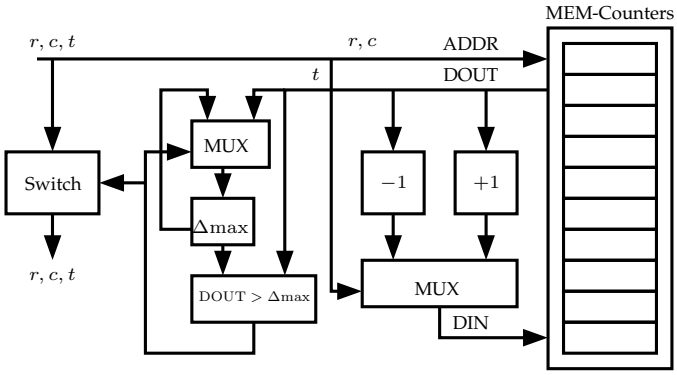

Fig. 4: Stochastic-based computing circuit for max pooling.

In this work, because all neural signals are encoded as random sample streams, fortunately, various pooling functions can be performed stochastically, i.e., manipulating random sample in simple ways. For the average pooling, its stochastic computing turns out to be quite straightforward. All we need to do is to aggregate all streams of random samples located within each pooling subzone. Note that there is no change needed for all random samples and only their indices need to be changed by modular operations. We now provide a detailed description of our stochastic max-pooling algorithm shown in Fig. 4. Specifically, at each location (r, c), we maintain an accumulative counter $\texttt{CT}[r][c]$. For each subzone where we perform the `max` operation, we have an aggregated counter $\Delta_{\max}$, which records the largest index value for all random samples seen so far in this subzone. Finally, three assisting functions $\texttt{sign}(s) = t$, $\texttt{idx}_\texttt{x}(s) = r$, and $\texttt{idx}_\texttt{y}(s) = c$ reads out the sign, x-index, and y-index of the random sample $s(r, c, t)$, respectively. The key idea of Fig. 4 is to output the correct number of random samples corresponding to the location (r, c) with the largest probability value.

4 Stochastic-Based CNN Architecture

Our proposed architecture is based on the stochastic computing principles, where all CNN layers are performed stochastically as presented in Section 3. In the following, we propose an efficient hardware structure to implement a stochastic-based CNN.

Our stochastic-based CNN architecture attempts to achieve both high scalability, therefore capable of handling extremely big networks, and high efficiency in hardware usage and off-chip memory communications while consuming low power. Therefore, this proposed architecture can facilitate the hardware realization of CNNs with FPGA devices while considering hardware resources and memory bandwidth limitations.

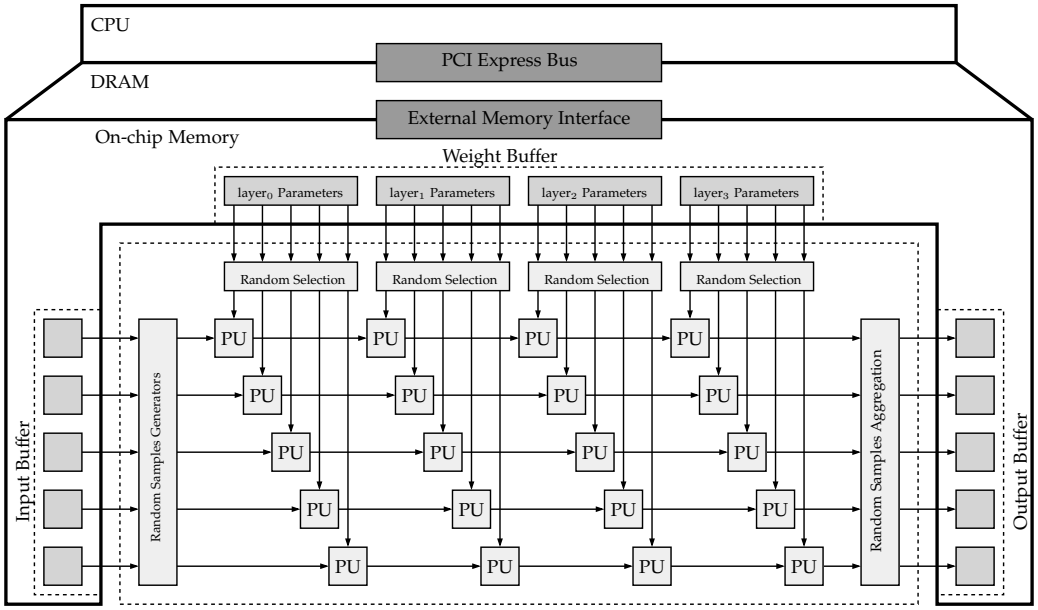

Fig. 5: Scalable stochastic CNN architecture, where PUs are stochastic processing units.

In the stochastic-based CNN implementation, shown in Fig. 5, the input image data is read from the external memory block by block, each is 2-D $N \times N$ vector, and cached in on-chip buffers. In order to maintain the relatively high speed of our proposed computation compared to the data transfer time, we use double data buffers. While one date buffer is operated upon, the other data buffer will undergo the data loading process. Random number generators are used to transform the real values of a 2-D vector, representing the input image or the convolution kernels, into random samples, whose 2-variate probability density function mirrors the 2-D vector values.

The proposed stochastic-based realization of CNN accelerator is highly scalable. A massive number of random-sample streams are implemented and organized in parallel to improve the computation performance. These stochastic processing unit lines are completely independent and construct from stochastic processing units. The structure of a stochastic processing unit is shown in Fig. 6. Each PU receives random samples representing a feature map and its corresponding parameters. Then, perform stochastic convolution followed by stochastic activation and pooling, if available for this layer. The resultant random samples are pushed to the next layer. The outcomes are aggregated at the end to accumulate random samples. Finally, the real CNN outputs numbers are extracted from the resultant random samples.

This scheme significantly reduces the hardware cost compared to the conventional processing unit. It provides a high scalable structure by streaming random samples through a sequence of stochastic processing units. The number of parallel streams in processing is determined by the hardware resources availability. We can realize each stream on a single FPGA device since there is no intermediate connection between them. At the end, all streams need to be combined in order to get the final results. The stochastic-based selecting circuit is used to determine the number of random samples to be generated for each kernel based on its coefficients, and to generate random samples r_k, c_k to be convolved with the random samples r_i, c_i that are coming from the previous layer. Kernel buffers are used to avoid the delay in reading from off-chip memory. A synchronization signal is used between layers to specify this random sample stream corresponding to which feature map, because each feature map needs to be convolved with specific kernels.

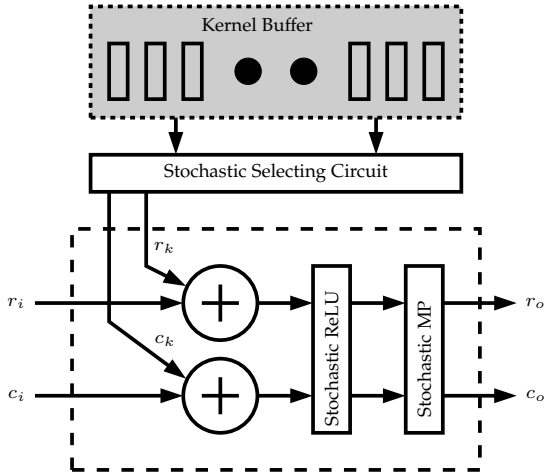

Fig. 6: Structure diagram of the stochastic processing unit for a convolution layer with activation and pooling.

In CNNs, fully connected layers can be converted to convolution layers [40] leading to fully convolutional neural networks. For example, ConvNet [37] has three fully connected layers. The first one is converted to a 7×7, which is the size of inputs to the fully connected layer, convolution layer, while the last two are converted to 1×1 convolution layer each. Therefore, the whole proposed circuit deals with random samples only since all layers are performed stochastically. At the output, channels of random sample streams, one for each class, are used to determine the class

score map. Thus, one counter is required for each channel to extract the real value from its corresponding random samples, i.e. 1000 for ConvNet [37]. Then, they are fed to a 1000-way soft-max classifier which produces a distribution over 1000 class scores. In the following, we provide more circuit design and implementation details. Specifically, we present how to generate random samples that follow any given distribution. Then discuss reducing the number of connections between layers and achieve streaming realization. Also, we show the ability of our proposed scheme in reducing the on-chip parameters requirement.

4.1 Random Numbers Generation

Random number generators are used to generate random samples follow the distribution of the 2-D input image or kernel coefficients. Each random sample is a three-tuple, (r, c, and t), where r and c denote the x- and y-index of the matrix and t denotes the sign of the corresponding matrix entry. The entry absolute value determines the probability that the random sample (r, c) falls within a specific index range (x, y), while the negative sign is used to determine the impact of the sample on the ultimate output value and also used for the stochastic ReLU and max pooling as presented in Section 3.

Given a 2-D $k \times k$ vector, for example kernel coefficients $\mathbf{W_{x,y}}$, we would like to generate an ensemble of random samples of the random variable r, c. These random samples will be exclusively drawn from the set $\{0, 1, \cdots, k-1\}$ and satisfy $\mathrm{P}(r = x, c = y) = \frac{\mathbf{W_{x,y}}}{\sum_{i=0}^{k-1}\sum_{j=0}^{k-1}\mathbf{W_{i,j}}}$ for any $x, y \in \{0, 1, \cdots, k-1\}$. Obviously, to achieve high efficiency, we will avoid calculating all probability values $\frac{\mathbf{W_{x,y}}}{\sum_{i=0}^{k-1}\sum_{j=0}^{k-1}\mathbf{W_{i,j}}}$. Additionally, we require that the total number of random samples will be solely determined by the user. In other words, at any point of random number generation, all samples that have been generated should faithfully follow the given PDF, which is called the ergodic property in statistics.

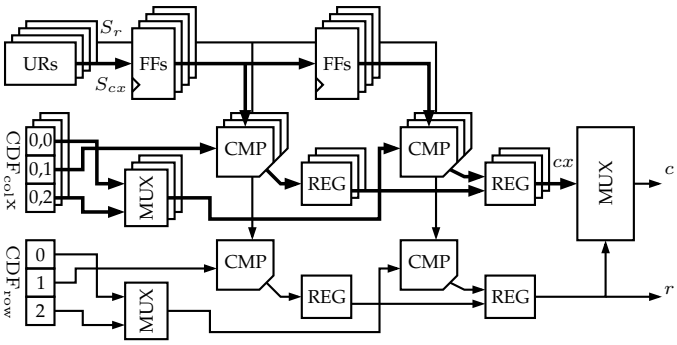

Fig. 7: Random number generator circuit when $k = 3$, where URs generate a uniformly distributed random numbers.

The block diagram of the proposed random sample generator is depicted in Fig. 7. For each row in the 2-D vector, we generate the cumulative distribution function, $CDF_{col}\mathrm{x}(y) = \sum_{i=0}^{y}\mathbf{W_{x,i}}$. Then, we uniformly draw a random sample S_{cx} for each $CDF_{col}\mathrm{x}$ from its corresponding closed range $[1, CDF_{col}\mathrm{x}(k)]$. Finally, we locate the particular segmented range $[CDF_{col}\mathrm{x}(i), CDF_{col}\mathrm{x}(i+1)]$ that contains S_{cx}, i.e., $CDF_{col}\mathrm{x}(i) < S_{cx} \leq CDF_{col}\mathrm{x}(i+1)$, where $i \in \{0, 1, \cdots, k-1\}$. Each one of the k parallel comparison circuits generates a column random sample; therefore, in parallel with this, we generate the cumulative distribution function of the rows, $CDF_{row}(x) = \sum_{i=0}^{x} CDF_{col}\mathrm{i}(k)$. Another uniformly distributed random sample S_r is generated and compared with the entries of CDF_{row}, as stated above. Then, the output of this circuit will be used as the row random sample and as the Mux control signal to determine which c_x will be used as the column random sample. Note that, in the scheme depicted in Fig. 7, binary search tree is used in order to find the correct range index; therefore, we need to perform $\log k$ comparisons. Fortunately, all comparisons are performed in a pipelined fashion, which significantly improve the throughput.

4.2 Reducing the Connectivity Between Layers

Conventionally, the output feature map of a specific layer equals the accumulation results of convolving P input feature maps with P different kernels. This step is functionally equivalent to a 3-D convolution, where the height and width are the kernel dimensions, $k \times k$, and the depth is P as in the following equation:

$$Q_i = P_1 * k_1 + P_2 * k_2 \cdots + \cdots + P_p * k_p, \qquad (1)$$

where P_i and k_i denote the i-th feature map and kernel, respectively. Therefore, Equation 1 can be realized stochastically, as presented in Section 3, by randomly selecting each input feature map, or its corresponding random samples stream. The kernels' coefficients are used to determine the number of samples that need to be selected from a specific feature map.

Fig. 2 shows the structure of the first convolution group of ConvNet [37] for one channel. In our proposed architecture, all the operations are performed stochastically as presented in Section 3. Therefore, a stream of k^* row-column (r-c) random samples, follows the distribution of the input image, is generated by the random number generator circuit shown in Fig. 7 and flows through the convolver circuits in the first layer. Each convolver circuit receives a stream of random samples follow the distribution of its corresponding kernel, and uses a pair of adders to add the image's r-c random samples with the kernel's r-c random samples and produce the addition stream of random samples. After applying the stochastic ReLU and stochastic pooling presented in Section 3, if available, a new stream of random samples will flow to the next layer. In the second layer, P streams of random samples, coming from the previous layer, need to be treated with P separate kernels stochastically to produce one output feature map. Therefore, Pk^* random samples need to be processed in the second layer for each output feature map, while the required number of samples to achieve no more than d error with $1 - \alpha$ confidence level, as we will present in Section 5, is k^* random samples for each extracted feature map. This problem gets worse as we go deeper in the network. Therefore, we propose a stochastic selection scheme to randomly select k^* samples out of Pk^* samples.

Fig. 8(a) depicts the structure of naively selecting one of the output random stream, where the selection circuit is placed after the processing unit in the second layer. There

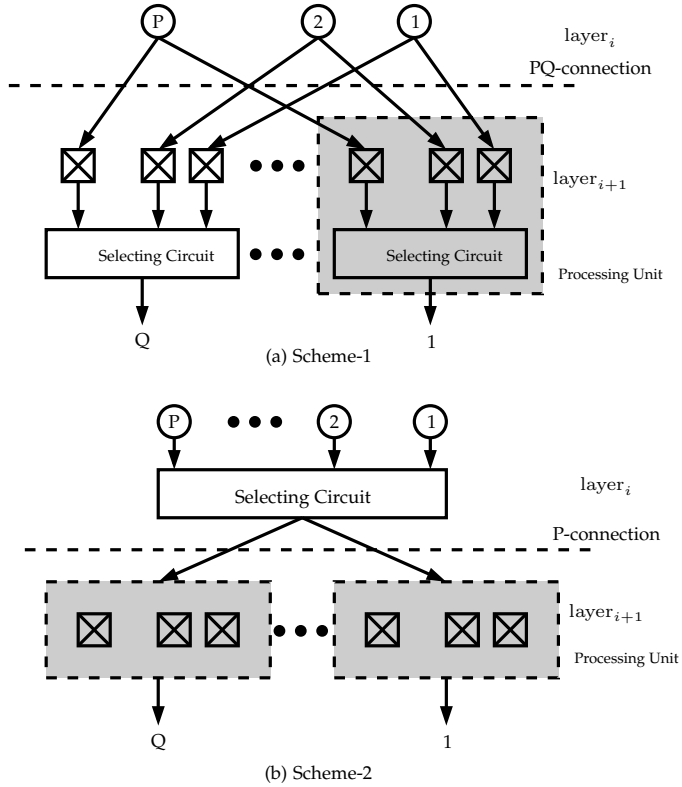

Fig. 8: Stochastically selecting S random samples out of PS random samples. (a) Scheme-1: naive selecting circuit (b) Scheme-2: proposed selecting circuit

are two issues with this structure: First, the connections between layers is as complicated as the one in the conventional structure, which has PQ connections. Second, $P-1$ pairs of adders will be redundant in each processing unit. They perform stochastic addition, but the selection circuit only selects one of them. Therefore, the structure shown in Fig. 8(b) is used to solve these problems. In this structure, the selection circuit is moved to the previous layer, where only one stream will flow from one layer to another. The advantages of using the second scheme is reducing the connection complexity between layers, where only Q connections are required. Therefore, if layer_i is realized on one FPGA device and layer_{i+1} on another FPGA device, then only one connection channel is required to connect them. The second advantage of using this structure is reducing the hardware usage, where only one pair of adders is required for each processing unit.

4.3 Random Selection Process

The accuracy of an output feature map or its corresponding distribution is determined by two factors. First, the number of random samples k^* needed to approximate the PDF, which will be discussed later in Section 5. Second, these random samples need to be thoroughly mixed, since they come from several input feature maps, to more accurately emulate the process of each processing unit. In other words, how to efficiently select k^* random samples out of Pk^* random samples coming from P input feature maps. Note that the PDF of feature maps will not be extracted till the end, as mentioned above, to keep the streaming of random samples through the network and avoid the need for PDF extraction units. However, we need to make sure that the feature maps, resulting from stochastic-based computing circuit after every layer, are corresponding to their equivalent feature maps in the conventional CNNs. The above-discussed scalable architecture demands us to perform such stochastic mixing in segments. Such stochastic mixing consists of two layers of mixing. Between input feature maps and kernels, random samples are generated and then added individually. In addition, all output feature maps of a layer are also mixed through selecting one of them stochastically using the selection circuit.

Fundamentally, there are two ways of approximating a PDF through combining different streams of random samples, where the stream results from convolving two random variables stochastically by the addition process. In the first method, because different streams may account for different probability mass function values, in theory, the probability of choosing different streams should be linearly proportional to their probability masses in order to obtain the overall accurate PDF. In contrast, the second method chooses each stream with an equal probability according to a uniform probability function. The different probabilistic mass value of each stream will be compensated with different number of random samples generated for each stream. Specifically, given a $k \times k$ kernel coefficients W_p, we define the kernel weight B_p as $\sum_{i=0}^{k-1}\sum_{j=0}^{k-1} W(i,j)$, where p denotes the index value of p-th kernel that will be convolved with the p-th stream of random samples corresponding to the p-th input feature map. Also, we choose constant values S_l and B_l to be the reference number of sufficient random samples and the reference probabilistic mass function value, respectively, where l is the layer sequence. Therefore, each time a stream P_p is uniformly chosen, its necessary random samples will be determined by a simple formula $S_p = \left(\frac{B_p}{B_l}\right) S_l$. In this paper, we select the second method for our implemented hardware prototype although it requires a multiplier because of its relative low hardware complexity and its ease of circuit implementation. Finally, the above stochastic mixing process can be modeled using Stochastic Differential Equations (SDEs) and its algorithm convergence, equivalent to the study of the stochastic stability of a Virtual Stochastic Dynamic System (VSDS), can be established by leveraging the techniques of Lyapunov stability as in [41].

4.4 On-chip Parameters Optimization

From the above discussion, we can see that not all the kernels in a specific layer are needed to be available on-chip simultaneously. For example, as presented in Fig. 9, when W_1 of the first layer is under processing, the other kernels of the same layer,$\{W_2 \cdots W_P\}$, are on hold. Where the stochastic selecting circuit of the first layer is used to determine the number of samples need to be generated from W_1's RNG circuit to be processed with the stream coming from the input image. At the same time, the output stream results from convolving the input image with W_1 represents P_1 feature map of the second layer. This feature map is convolved only with the first kernel of each processing unit, $\{W_{11}, W_{21}, \cdots W_{Q1}\}$. Therefore, only

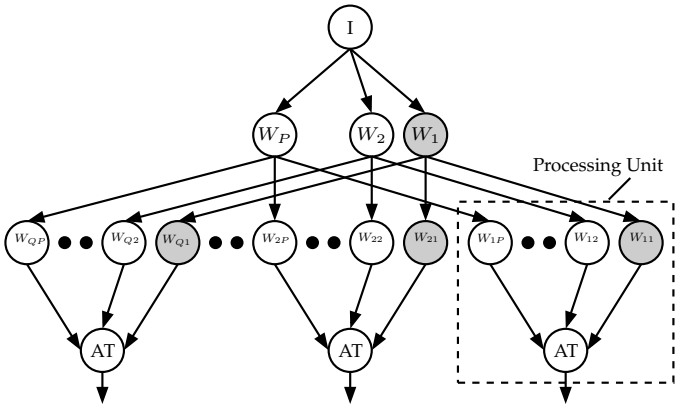

Fig. 9: Active processing units in CNN, and the required on-chip parameters.

these kernels are required to be processed with P_1's stream. Similarly, the stochastic selecting circuit of the second layer determines the number of samples corresponding to each kernel, $\{W_{11}, W_{21}, \cdots W_{Q1}\}$, based on their coefficients to be processed with P_1's stream. Thus, only one of them has to be on-chip, as in the first layer. This same principle can be applied to all other streams. Each channel of random stream computing units can be used to process a separate image to achieve high performance computation and compete GPU realization. Since same weight parameters are used for each image, the number of parameters needs to be available on-chip is reduced.

5 Error Analysis

The success of our stochastic-based CNN hinges on the fact that we can accurately and efficiently generate an ensemble of random samples that represent any given probability value. One critical question to ask is how many random samples are enough to achieve any required computing efficiency. In this paper, the most important building block of SCNN is adding two independent random variables and extracting the probability density of its resulting sum. In the following, we derive the probabilistic error bound in two steps. First, we investigate the required number of random samples in order to precisely extract the desired probability density function. Second, we obtain the relationship between the total number of random samples and the overall accuracy of our probabilistic convolution.

Formally, let $\mathbf{X} = \{x_1, x_2, \cdots, x_N\}$ denotes a given N-sized input vector. To approximate $p_i = \frac{x_i}{\sum_{i=1}^{N} x_i}$, which has probability at least $1-\alpha$ of being no further than d from the accurate value, the necessary random sample size requires no more than

$$k^* = \frac{z^2 p_i(1-p_i)}{d^2}, \tag{2}$$

Furthermore, the required random sample size to extract a probability density function $f_{\mathbf{X}}$ consists of a sequence of p_is, where $i = 1, 2, \cdots, N$. This is an equivalent problem of finding sample size for estimating several proportions simultaneously. In terms of the required sample size, the worst-case scenario occurs when the combination of N proportions that give the maximum probability of a sample for which at least one of the sample proportions was unacceptably far from the corresponding population proportion [42]. Considering Equation (2), because $0 \leq p_i \leq 1$, $p_i(1-p_i)$ takes its maximum value $1/4$ when $p_i = 1/2$. Therefore, to approximate $f_{\mathbf{X}}$ with a probability at least $1-\alpha$ of being no further than d from the accurate value for each p_i, where $i = 1, 2, \cdots, N$, the necessary random sample size requires no more than

$$k^* = \frac{z^2}{4d^2}, \tag{3}$$

where d is the absolute error and z is the upper $\alpha/2$ point of the normal distribution. Table 1 lists some numerical results according to Equation 3.

TABLE 1: Random sample size k^* for approximating any p_i value within in distance $d = 0.05$ of the true value at confidence level $1-\alpha$.

$1-\alpha$	d^2k^*	k^*
0.50	0.44129	177
0.90	1.00635	403
0.95	1.27359	510
0.99	1.96986	788

6 Results and Analysis

In this section, we present the experimental results of our proposed stochastic-based CNN architecture and the comparisons with other FPGA-based CNNs. We used a Xilinx Virtex-6 FPGA device to implement our proposed SCNN. Since our proposed architecture is based on streaming random samples through a sequence of stochastic processing units (PUs), as presented in Section 4, we can implement several channels of random stream processing layers for a given amount of hardware resources. As such, all random samples will be combined at the end to determine the score map vector. For all the presented results in this section, we only use one channel of random samples streaming and everything is implemented on a single Virtex-6 LX550T FPGA chip. Also, we found that each PU needs $D/2$ kernel buffers to avoid memory access delay, where D is the depth of a CNN. To use the FPGA resources efficiently, our proposed architecture uses BRAMs to buffer on-chip data and to store the counters corresponding to the stochastic ReLU and max pooling. It uses DSP slices for pre/post processing and also in stochastic selecting circuit to determine the number of random samples for each kernel coefficients. Logic slices are used to realize all other components of the architecture.

To make our performance comparisons fair, we measure the total execution time for a given computing task using stochastic-based architecture and compare it to the equivalent conventional ones. Specifically, we calculate the required stochastic computations to perform convolutional network and find the total execution cycles for different accuracy levels. Subsequently, we convert the results to GOPS (Giga Operations Per Second), by dividing the CNN size over the execution cycles and multiply the result by the cycle period, and use it as the performance metric. Because different FPGAs have been used to realize CNNs, we use the performance density as another performance metric [22], to understand and compare hardware-efficiency

TABLE 2: Comparison with other large-scale Deep CNN implementations.

	Conventional CNN					Stochastic CNN		
	ISCA2010 [24]	FPGA2015 [22]	FPGA2016 [30]	FPGA2017 [43]	FPGA2018 [44]	TMSCS2016 [34]	Proposed	
FPGA	Virtex5 SX240T	Virtex7 VX485T	Zynq XC7Z045	Arria-10 GX 1150	Xilinx Virtex 690t	Virtex6 LX550T	Virtex6 LX550T	
Frequency (MHz)	200	100	150	150	150	200	200	
Power (W)	14	18.61	9.63	21.2	25	6.81	3.61	
CNN size (GMAC)	0.26	1.33	30.76	30.95	30.95	30.76	30.76	
$1-\alpha$						99%	99%	95%
Performance (GOPS)	16	61.62	136.97	645.25	570	72.76	581.93	899.13
Performance Density (GOPS/Slice)	4.3E-04	8.12E-04	26.1E-04	40.10E-04	32.57E-04	17.49E-04	135.4E-04	209.2E-04
Power Efficiency (GOPS/W)	1.14	3.31	14.22	30.44	22.8	10.68	161.19	249.06

of different CNN implementations. In this paper, we define the hardware density as the average GOPS per logic slice (GOPS/Slice). Note that the logic capability of each logic slice across different FPGA devices mostly remains the same, therefore this performance metric of hardware density conceptually indicates how computationally effective a given unit of hardware is being used. Intuitively, this computing density number is also closely related to the energy efficiency for computing.

In our proposed architecture, all processing units deal with stochastic samples, and there is no need to store intermediate data resulting from loop tiling technique and inter-layer connections, as in the conventional architectures. We only need to read input images at the first layer and store the output of the last layer. Most of the off-chip memory access will be performed to read CNN's weights, kernel coefficients. This can significantly reduce the overall power consumption of CNNs, which is typically dominated by data transfer and memory access operations. In contrast, for the conventional deterministic-based CNN architecture, a large amount of intermediate data needs to be stored and retrieved, which translates into huge numbers of read/write operations from/to off-chip memory. This consumes a significant amount of energy consumption even larger than their associated algorithmic operations. For example, the average number of intermediate data, in pixels, of a conventional CNN architecture that needs to be read/written from/to an off-chip memory when $P = 256$, $Q = 256$, and $N = 56$ is 3.67M pixel.

To benchmark the performance of our proposed stochastic-based CNN architecture in large scale, we choose the deep learning CNN configuration that wins the first place in image classification task of the 2015 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). It consists of five convolution layer groups and three fully connected layers. There are different versions of this model, e.g. VGG-11, VGG13, VGG-16, and VGG-19. The number of convolution layers in each group varies in each version. In this paper, we use VGG-16, pre-trained using MatConvNet MATLAB toolbox[1], as the standard model for all the presented results. The model constructs from 13 convolution layers followed by 3 fully connected layers. The size of all the convolution kernels is 3×3. Each convolution layer is followed by the rectification non-linearity, and each group has one max pooling layer. In all the following results, we adopted the Image-Net dataset to evaluate the performance of our proposed stochastic CNN, which includes 1.3M images for training dataset and 50K images for validation dataset divided into 1000 classes. For the stochastic-based CNN software implementation, we re-implemented all the CNN processing units following the stochastic-based algorithms presented in Section 3.3.

Table 2 compares our proposed Deep SCNN with existing FPGA-based architectures. It shows that ours outperforms other architectures in terms of performance, hardware resources utilization and power efficiency. The performance of SCNN mainly depends on the number of random samples $S = k^* \times featuremaps$, where k^* is the number of random samples required to achieve a specific computing accuracy for each feature map, as presented in Section 5. The results show that Deep SCNN achieves 1.58x more GOPS compared to the state-of-the-art CNN architecture [44] to achieve $d = 0.01$ with 95% confidence level. In terms of area efficiency, our proposed method achieves 3.35x more GOPS/Slice than the best architecture among the others. Finally, the last row in Table 2 presents the power efficiency of our implementation. It shows that SCNN achieves 10.92x more GOPS/W compared with the CNN in [44].

TABLE 3: Comparison with the most recently published conventional CNN architecture where ET is the execution time.

	ET	Energy	Energy Efficiency	Top-5
	ms	J	GOPS/J	%
FPGA2016 [30]	224.6	2.16	63.41	86.66
FPGA2017 [43]	47.97	1.02	632.60	within 2% less than [30]
FPGA2018 [44]	89.4	2.24	254.46	–
Proposed	52.9	0.19	3062.78	86.77

We also compared our proposed architecture with conventional FPGA-based CNN accelerators in terms of execution time, energy efficiency and accuracy loss. The results presented in Table 3 show that state-of-the-art conventional realization requires 1.70x more time and consume 11.79x more energy than the SCNN to finish all the computation. This makes the proposed method 12x more energy efficient than the most recently published paper. In terms of performance accuracy, the proposed method can achieve a frame rate at 18.90 fps with the top-5 accuracy of 86.77%, while the deterministic-based architecture can achieve a frame rate at 4.45 fps with the top-5 accuracy of 86.66% [30].

To validate the functionality of our proposed stochastic-based CNN, we present in Fig. 10 a comparison between the conventional deterministic-based CNN and the stochastic-based CNN with different accuracy levels, by changing the

1. http://www.vlfeat.org/matconvnet/

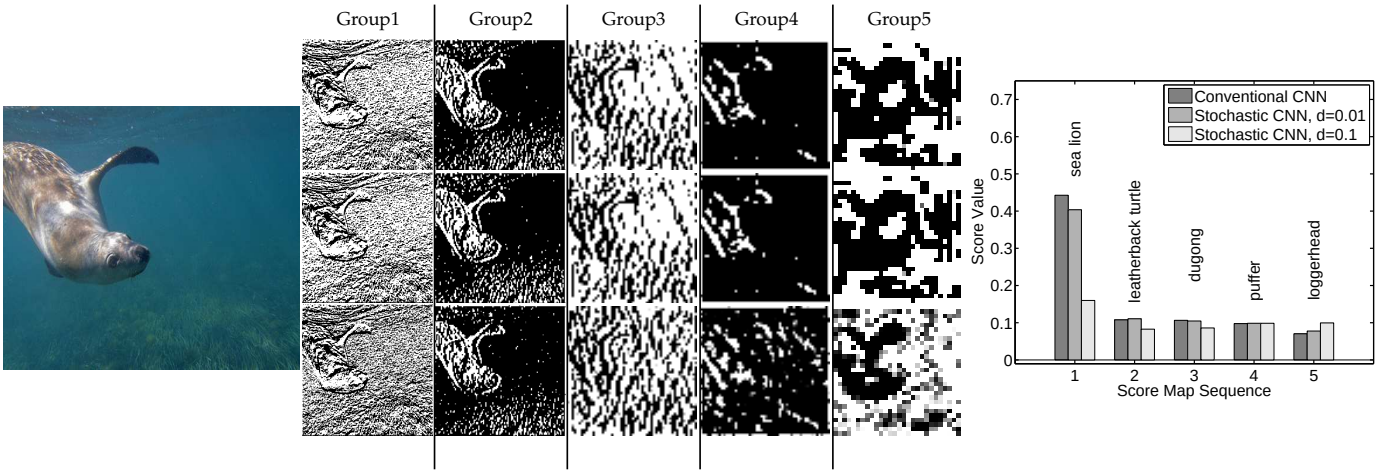

Fig. 10: Stochastic-based VGG-16 ConvNet Illustration example. The first row represents the conventional CNN results, the middle and last rows represent stochastic-based CNN results when $d = 0.01$ and $d = 0.1$, respectively.

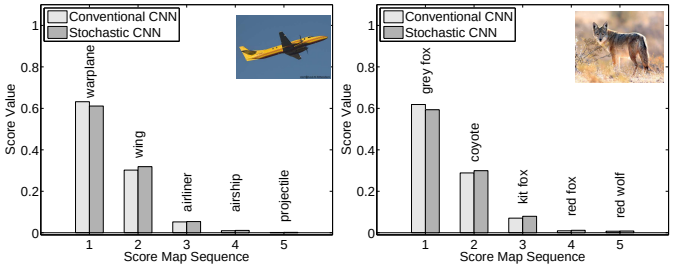

Fig. 11: Two ImageNet dataset test cases from VGG16 CNN.

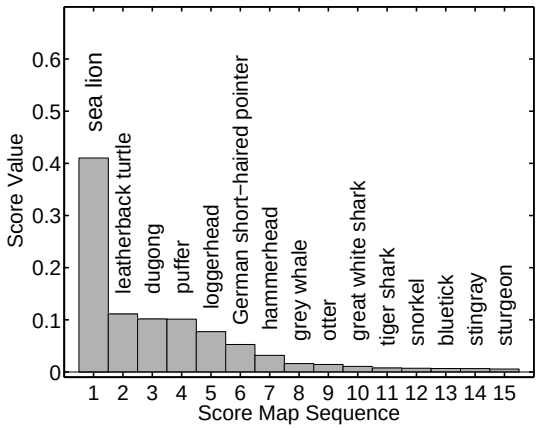

Fig. 12: The top 15 scores from the VGG-16 ConvNet score map vector.

number of random samples, to show the impact of random samples size towards the stochastic CNN computing accuracy. We have selected one output feature map from each convolution group in VGG-16 model and draw it. The first row shows the results when the conventional CNN is used, while the second and third rows show the stochastic-based CNN results when $d = 0.01$ and $d = 0.1$, respectively. Then, all resulting ranking scores are measured, which clearly show how the difference between scores increases with less random samples, especially for the top-1 score. We have also chosen two representative image recognition cases from ILSVRC 50,000 validation dataset. As shown in Fig. 11, in both cases our stochastic-based CNN can produce the correct image recognition results with a little difference in ranking scores. The main reason makes stochastic-based computing works for this kind of applications is that there is no golden output result needs to be achieved. As we see in Fig. 12, the output is a vector of scores, we selected the top 15 out of 1000, each corresponding to a specific class. The recognition is correct as long as the score of its class has the highest value among others. Top1 and Top5 are used to determine the accuracy of our proposed stochastic CNN. They represent the percentage of matching the input image to the highest rank label or to the first 5 highest rank labels within the predicted list for the whole validation dataset, respectively. Top1 and Top5 for conventional CNN are 65.99% and 86.87% respectively. The accuracy in SCNN can be controlled by the number of random samples, where our results have shown that SCNN can achieve 65.68% and 86.77% for Top1 and Top5 respectively.

7 Conclusions

In terms of computing performance and energy efficiency, our stochastic-based CNN realization can achieve significant improvements over the state-of-the-art deterministic CNN accelerators. Maybe one of the most novel aspects of our stochastic-based CNN is its capability to seamlessly improve computing accuracy incrementally by dynamically adjusting the number of random samples to be processed without making any changes to the computing hardware at run time. This feature can be quite essential in many mission-critical embedded applications. Moreover, we prove that stochastic-based computing can significantly improve the circuit scalability of a large-scale CNN, which makes multi-FPGA CNN implementation much more straightforward and scalable. Finally, the stochastic-based computing principle in our CNN implementation strategy

naturally lends itself to training CNNs with probabilistic dropping that can effectively avoid harmful data overfitting, thus opening up a new front and an elegant alternative computing methodology to CNN implementation. Our future work will focus on developing CNNs training technique based on the stochastic computing principle.

ACKNOWLEDGEMENT

This work was supported in parts by an ARO DURIP grant N16-22-6056, an NSF BRIGE grant ECCS-1342225 and an NSF CCF grant 1319884.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Sys. 25*, Curran Associates, Inc., 2012.

[2] D. Triantafyllidou and A. Tefas, "Face detection based on deep convolutional neural networks exploiting incremental facial part learning," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 3560–3565, Dec 2016.

[3] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014.

[4] W.-t. Yih, X. He, and C. Meek, "Semantic parsing for single-relation question answering," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, (Baltimore, Maryland), pp. 643–648, Association for Computational Linguistics, June 2014.

[5] Q. Zhang, Y. N. Wu, and S. Zhu, "Interpretable convolutional neural networks," *CoRR*, vol. abs/1710.00935, 2017.

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015.

[7] A. Mohamed, F. Seide, D. Yu, J. Droppo, A. Stolcke, G. Zweig, and G. Penn, "Deep bi-directional recurrent networks over spectral windows," in *IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU, Scottsdale, AZ, USA.*, pp. 78–83, 2015.

[8] Y. LeCun, "Deep learning convolutional networks," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, pp. 1–95, Aug 2015.

[9] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, (New York, NY, USA), pp. 5–14, ACM, 2017.

[10] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *ICANN*, 2014.

[11] W. W. L. Fung, *GPU computing architecture for irregular parallelism*. PhD thesis, University of British Columbia, 2015.

[12] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," *CoRR*, vol. abs/1602.04283, 2016.

[13] D. V. Negrov, I. M. Karandashev, V. V. Shakirov, Y. A. Matveyev, W. L. Dunin-Barkowski, and A. V. Zenkevich, "A plausible memristor implementation of deep learning neural networks," *CoRR*, vol. abs/1511.07076, 2015.

[14] B. Dally, "Power, programmability, and granularity: The challenges of exascale computing," in *2011 IEEE International Test Conference*, pp. 12–12, Sept 2011.

[15] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, Feb 2014.

[16] V. D. Suite, "Ultrascale architecture fpgas memory interface solutions v7.0," *Technical report, Xilinx*, April 2015.

[17] S. Mittal, "A survey of techniques for managing and leveraging caches in gpus," *Journal of Circuits, Systems, and Computers (JCSC)*, vol. 23, 2014.

[18] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, Nov 2016.

[19] T. Hamilton, S. Afshar, A. van Schaik, and J. Tapson, "Stochastic electronics: A neuro-inspired design paradigm for integrated circuits," *Proceedings of the IEEE*, vol. 102, pp. 843–859, May 2014.

[20] M. Alawad and M. Lin, "Stochastic-based multi-stage streaming realization of deep convolutional neural network," in *2017 18th International Symposium on Quality Electronic Design (ISQED)*, pp. 13–18, March 2017.

[21] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014.

[22] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, (New York, NY, USA), pp. 161–170, ACM, 2015.

[23] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 92–104, June 2015.

[24] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *SIGARCH Comput. Archit. News*, vol. 38, pp. 247–257, June 2010.

[25] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *CVPR 2011 WORKSHOPS*, pp. 109–116, June 2011.

[26] S. A. Dawwd, "The multi 2d systolic design and implementation of convolutional neural networks," in *Electronics, Circuits, and Systems, IEEE 20th International Conference on*, pp. 221–224, Dec 2013.

[27] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pp. 13–19, Oct 2013.

[28] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "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*, ASPLOS '14, pp. 269–284, ACM, 2014.

[29] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *CoRR*, vol. abs/1604.03168, 2016.

[30] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, (New York, NY, USA), pp. 26–35, ACM, 2016.

[31] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating deep convolutional networks using low-precision and sparsity," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2861–2865, March 2017.

[32] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, (Piscataway, NJ, USA), pp. 1–13, IEEE Press, 2016.

[33] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[34] M. Alawad and M. Lin, "Stochastic-based deep convolutional networks with reconfigurable logic fabric," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, pp. 242–256, Oct 2016.

[35] F. Nasse, C. Thurau, and G. A. Fink, "Face detection using gpu-based convolutional neural networks," in *Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns*, CAIP '09, (Berlin, Heidelberg), pp. 83–90, Springer-Verlag, 2009.

[36] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "Cnp: An fpga-based processor for convolutional networks," in *International Conference on Field Programmable Logic and Applications*, pp. 32–37, 2009.

[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[38] C. M. Grinstead and J. L. Snell, *Introduction to Probability*. AMS, 2003.

[39] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (J. Frnkranz and T. Joachims, eds.), pp. 807–814, Omnipress, 2010.

[40] E. S. Jonathan Long and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.
[41] J. Chen and V. K. N. Lau, "Convergence analysis of mixed timescale cross-layer stochastic optimization," *CoRR*, vol. abs/1305.0153, 2013.
[42] S. Thompson, *Sampling*. CourseSmart, Wiley, 2012.
[43] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, (New York, NY, USA), pp. 45–54, ACM, 2017.
[44] J. Shen, Y. Huang, Z. Wang, Y. Qiao, M. Wen, and C. Zhang, "Towards a uniform template-based architecture for accelerating 2d and 3d cnns on fpga," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, (New York, NY, USA), pp. 97–106, ACM, 2018.

Mohammed Alawad is a Research Associate in the Health Data Sciences Institute and the Biomedical Sciences, Engineering and Computing Group at the Oak Ridge National Laboratory. He received his PhD degree in Computer Engineering from the University of Central Florida, Orlando, Florida, in 2016. His current research interests include stochastic computing, deep learning, natural language processing, energy-efficient and high-performance computing.

Mingjie Lin Mingjie Lin is currently an Associated Professor at ECE department of University of Central Florida. His research interests are focused on FPGA memory optimization in High-Level Synthesis and novel computing techniques for high-dimensional robotic control.