This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Double MAC on a DSP: Boosting the Performance of Convolutional Neural Networks on FPGAs

Sugil Lee*†, Daewoo Kim*, Dong Nguyen* and Jongeun Lee*†
*School of ECE, UNIST, Ulsan, South Korea
†Neural Processing Research Center, Seoul National University, Seoul, Korea
jlee@unist.ac.kr

***Abstract*—Deep learning such as Convolutional Neural Networks (CNNs) are an important workload increasingly demanding high-performance hardware acceleration. One distinguishing feature of deep learnng workload is that it is inherently resilient to small numerical errors and works very well with low precision hardware. Thus we propose a novel method, called *Double MAC*, to theoretically double the computation rate of CNN accelerators by packing two multiply-and-accumulate (MAC) operations into one DSP block of off-the-shelf FPGAs. There are several technical challenges, which we overcome by exploiting the mode of operation in the CNN accelerator. We have validated our method through FPGA synthesis and Verilog simulation, and evaluated our method by applying it to the state-of-the-art CNN accelerator. We find that our Double MAC approach can increase the computation throughput of a CNN layer by twice. On the network level (all convolution layers combined), the performance improvement varies depending on the CNN application and FPGA size, from 14% to more than 80% over a highly optimized state-of-the-art accelerator solution, without sacrificing the output quality significantly.**

***Index Terms*—Convolutional neural network, FPGA, SIMD (Single-Instruction Multiple-Data), reduced precision, DSP (Digital Signal Processing) block, MAC (Multiply-and-Accumulate).**

I. Introduction

As machine learning algorithms are getting more popular, there is an increasing demand for developing hardware accelerators for them. In particular deep neural networks such as Convolutional Neural Networks (CNNs) have multiple traits that make them very attractive for hardware acceleration, such as high structural regularity, high computational complexity, and yet wide applicability and high recognition performance. FPGAs are one of the most preferred platforms due to their high flexibility and at the same time high parallelism. Hence much effort has been made to create better CNN accelerators on FPGAs [2]–[5].

A unique option available to hardware implementations of DNNs is the flexibility in data width of arithmetic operations. GP-GPUs, for instance, have long provided only two options—either single-precision or double-precision floating point—since integer arithmetic on modern GP-GPUs has zero or negative performance advantage. Recently half-precision was introduced on some select models [6], but this is a one-time change and not cutomizable by user. By contrast an ASIC (Application-Specific Integrated Circuit) implementation can choose whatever precision sufficient for the target CNN application. As recent work suggests that 8-bit fixed-point is often enough for inference, even for deep CNNs [7], there is a good opportunity to increase performance for free by using lower precision without affecting output quality.

FPGAs, too, have the flexibility, and using reduced precision means potentially higher throughput on the same FPGA. In practice, however, since most arithmetic operations are implemented using DSP blocks, and DSP blocks, too, support only a limited set of precisions, it is not easy to achieve higher performance through reduced arithmetic precision. For example, the DSP block of Xilinx FPGAs, DSP48E1, can perform a 25x18-bit multiplication only [8], and there is no way to perform two 8x8-bit multiplications simultaneously on the same DSP block for higher throughput.

This paper is about how to turn an ordinary DSP block of an off-the-shelf FPGA device into a 2-way SIMD (Single-Instruction Multiple-Data) MAC (Multiply-and-Accumulate) unit, that can deliver 4 ops/cycle by performing two multiply-and-add operations simultaneously with reduced data width. Though we evaluate our technique for a Xilinx Virtex-7 FPGA only, our method is generic and applicable to other FPGAs with similar hardware DSP blocks. Our technique does not require any change in the FPGA fabric itself.

Realizing SIMD *add* on a DSP unit is trivial and it is already supported [8]. A SIMD *multiply* using LUTs (Look-Up Tables) is also trivial. The challenge is how to realize *SIMD multiply on a DSP block* of an FPGA. Using DSP blocks is important, since most DNN implementations on FPGAs rely on DSP blocks for MAC operations [2], and therefore being able to perform two MACs using one DSP block essentially means free 2X improvement in computation throughput. Further, though it is possible to increase throughput by other means (e.g., implementing additional MACs with LUTs [4]), our method is orthogonal to them.

Without native SIMD multiplication support on DSP blocks, we must create *virtual SIMD lanes* inside one DSP block, making sure that data on each lane do not collide with each other. We must also take care of sign bits and overflow detection, all within one DSP block. Not only is this far from straightforward, a simple analysis reveals that a general SIMD

A preliminary version of this paper appeared in [1].
This research was supported in part by Samsung Advanced Institute of Technology, KIST Institutional Program (Project No. 2E27810-18-P034), and Free Innovative Research Fund (1.180037.01) of UNIST, and in part by Nano-Material Technology Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (2016M3A7B4909668).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

multiply on one DSP block is impossible without modifying hardware. Our technique does not require any change in the FPGA fabric itself, but uses additional resources such as LUTs and FFs (Flip-Flops).

We make the following contributions in this paper. First we define a special class of SIMD multipliers, tailored for the kind of MAC operations found in convolutional layers of CNNs (Convolutional Neural Networks). Specifically, our Double MAC requires that the multiplications of a SIMD multiply share a common operand, viz., $A \times C$ and $B \times C$ instead of $A \times C$ and $B \times D$, and that the common operand, C, be an unsigned number. Second, with this restriction we show that it is possible to design a 2-way SIMD multiplier-and-adder within one DSP block with little overhead. Third, we demonstrate that despite its inherent restrictions our Double MAC architecture can be successfully incorporated into convolution layers of a CNN accelerator if input activations are unsigned numbers, which is typically the case as deep CNNs use the ReLU (Rectified Linear Unit) activation function. Fourth, for convolution layers whose input activations are *signed* numbers, we present a method to convert convolution layers of signed input into that of unsigned input. Finally, to compensate for the reduction in arithmetic precision, we present a shift-only (i.e., cost-free) scaling scheme for feature maps and weight parameters, and demonstrate that our Double MAC based CNN implementations can achieve high enough accuracy even for large, real-life CNNs thanks to scaling.

In this paper we assume that only inference, as opposed to training, of a CNN is done on an FPGA. We also assume that the CNN accelerator accelerates convolution layers only, which account for the vast majority of computation.

We validate our method through Verilog simulation and FPGA synthesis, and evaluate our method by applying it to one of the state-of-the-art CNN accelerator designs [2]. We demonstrate that our Double MAC can double the computation throughput at the MAC operation level, and also at the MAC array level when given the same number of DSP blocks, as compared to the previous state of the art. Our scheme does use more LUTs, but is far more efficient in terms of GOPS per LUT, as compared to synthesizing additional MACs using LUTs (e.g., [4]). On the network level, i.e., with all convolution layers combined, our method can generate performance improvements that range, depending on the CNN hyper-parameters and FPGA size, from 14% to more than 80% over the highly optimized state-of-the-art accelerators, designed for two large, real-life CNNs [9], [10], without sacrificing the output quality significantly, which is in part thanks to our scaling scheme.

The rest of the paper is organized as follows. After briefly reviewing related work and CNN accelerators in Section II, we present our Double MAC architecture in Section III, and its use in a state-of-the-art CNN accelerator in Section IV. We present our experimental results in Section V and Section VI concludes the paper.

II. Background and Related Work

A. CNN Accelerator

Convolutional layers account for the majority of the computation of a CNN evaluation. A convolutional layer takes as input a number (N) of matrices called *input feature maps*, and generates a number (M) of matrices called *output feature maps*. The computation of each output feature map (Y_m) involves the summation of N 2D-convolutions between each of the input feature maps (X_n) and each of the N weight parameter matrices ($W_{m,n}$). Ignoring bias addition, $Y_m = \sum_n^N W_{m,n} * X_n$, where $*$ represents convolution.

Naturally the repetitions along the M, N dimensions have been the source of parallelism exploited by multiple hardware accelerators recently [2], [11]. In particular, a very recent CNN accelerator developed for FPGAs [2] is based on a 2D array of multipliers and adders, as illustrated in Fig. 4(a). This MAC array consumes T_N inputs and generates T_M outputs simultaneously, and performs $T_N T_M$ MAC operations per cycle sweeping through the entire input feature maps and output feature maps in a manner similar to loop tiling. Finding the best values for T_N and T_M can be done trivially using an exhaustive search, which may also consider other parameters such as buffer size parameters to maximize data reuse, since in general the input/output feature maps cannot fit in the on-chip memory of an FPGA.

Our proposed Double MAC architecture is applicable to other CNN accelerators as well. However, to demonstrate the applicability of our technique and evaluate performance improvement in the most realistic setting, we use the accelerator architecture of [2], with a small enhancement to maximize the compute density.

B. Related Work

CNNs are known for their computational intensiveness, and many accelerators have been proposed for ASIC [11]–[13] as well as FPGA targets [2], [4], [5], [7], [14]. While ASIC implementations generally have lower cost and higher energy efficiency, they have no flexibility to support different accelerator architectures. In addition to being flexible, FPGA implementations also benefit from recent C-based design flows such as high-level synthesis (e.g., [2]) and OpenCL (e.g., [7]), which can help reduce time-to-market further. There are many other accelerator architectures that are not specifically targeted for one or the other (e.g., EIE [15]).

CNN accelerator architectures targeting FPGAs are typically built around a MAC array, and depending on how the MAC array is used to perform the computation of a convolution layer, different architectures may exist. Specifically, the computation of a convolution layer involves a number of MAC operations arranged in a six-deep nested loop [16]—two for the input feature map (N) and the output feature map (M), two for the row/column of an image (let R and C denote the number of rows and columns of an image), and two for the row/column coefficients of the convolution filter. Previous work on CNN accelerators explores different ways such as parallelizing along M-N loops [2], [4], M-R loops [5], M-C loops [5], and M-R-C loops [14]. A recent study provides a design space

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

exploration methodology for different parallelization schemes [16]. Our work is based on the parallelization along M-N loops, which is the same as [2], [4].

Early work on CNN acceleration used floating-point precisions (e.g., [2]), but recent designs use fixed-point precisions (e.g., [4], [7]). For inference as opposed to training, even 8-bit fixed-point is shown to be enough for some large CNNs [17].

Recently, reduced-precision CNN models have become a topic of great interest due to their implementation-friendly nature such as low power and low cost. For example, in the IBM TrueNorth processor [18], weight parameters are permitted to have one of four possible values. Authors in [19] presents a learning method for TrueNorth. Researchers are exploring even binarized neural networks (e.g., [20]) where weight parameters are either 1, 0, or -1. Authors of [21] attempt to analyze the impact of quantization in CNNs. A slightly different approach is to use approximate multipliers [22], [23].

On the other hand, our CNN accelerator is different from approximate computing-based ones (e.g., [23]–[26]). Approximate computing-based accelerators take advantage of the resilience of machine learning algorithms by using approximate but more energy-efficient functional units such as approximate adders and approximate multipliers, and thereby gain in cost and energy efficiency. Our MAC array produces exact results for the given input, and there is no truncation or rounding introduced by our MAC array, which is why we can use Caffe [27] to simulate our CNN accelerator.

There are several approaches that implement FPGA-based SIMD processors [28]–[31]. Most of those processors consist of an array of processing elements (PEs) that operates in a SIMD fashion, thus different from our definition of SIMD, which is really multi-word instruction. The work in [31] proposes a 2D convolution processor which implements a SIMD *convolver*. A 16-bit convolution operation is split into two simultaneously 8-bit convolutions by operating on the operands subwords. Our approach is completely different, since we execute two simultaneous operations on a single compute unit.

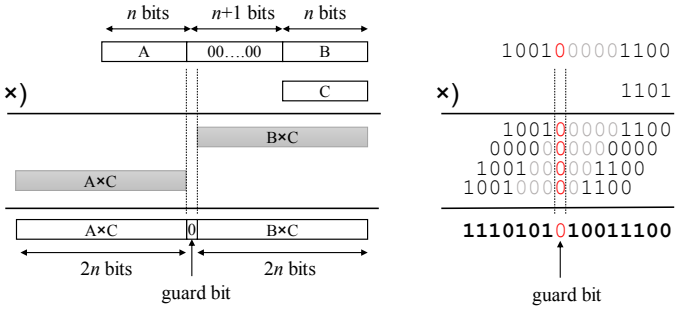
Our design supports two concurrent signed-unsigned multiplication with just one signed multiplier. Some correction steps are required since the second lane of our Double MAC is not able to recognize signed operation. A similar method to achieve correct signed-signed multiplication with unsigned multiplier is mentioned in [32].

III. Our Proposed Double MAC Architecture

We now present our double MAC architecture, which supports (i) SIMD multiplication and (ii) accumulation of many multiplication results. First we see how SIMD multiplication can be done using only one hardware multiplier in one cycle for both unsigned and signed numbers, and then see how to support accumulation of arbitrary number of operands.

A. SIMD Multiplication of Unsigned Numbers

Let us first consider the case where all the operands are unsigned. Fig. 1 illustrates how a single multiplier can perform



(a) n-bit SIMD unsigned multiplier (b) Example: 9×13 & 12×13

Fig. 1: How 2-way SIMD multiplication works (unsigned case).

two unsigned multiplications with a common operand simultaneously, i.e., $A\times C$ and $B\times C$. For this to work, two conditions must be satisfied. Let n be the width of each operand. First, the output register must be at least $4n$-bit wide. Second, the two operands in one of the inputs must be separated by at least n bits.

For accumulation to work without overflow, we need at least one *guard bit* as illustrated in the figure. During the accumulation the same guard bit can be used to *detect* any carry out of the lower $2n$ bits. On detection, the carry bit is cleared immediately and the number of carry-out events is counted separately using a small counter. The value of the counter is added separately once all accumulation is finished. Thus to perform two n-bit multiplications in a SIMD fashion we need one $(3n+1)\times n$-bit multiplier. For example, with the 25x18-bit multiplier of a DSP48E in Xilinx FPGAs, n can be at most 8.

B. SIMD Signed-Unsigned Multiplication

Let us consider how to perform a signed-unsigned multiplication using an unsigned multiplier. Let $B = b_{n-1}\cdots b_1 b_0$ be an n-bit signed integer, and C an n-bit unsigned integer. Recalling $-2^{n-1} = 2^{n-1} - 2^n$, the product $B\times C$ can be computed as follows, where $\hat{B}$ represents the value of B interpreted as an unsigned number.

$$\begin{aligned} B\times C &= (-b_{n-1}\cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i\cdot 2^i)\cdot C \\ &= (b_{n-1}\cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i\cdot 2^i)\cdot C - b_{n-1}\cdot 2^n\cdot C \\ &= \hat{B}\cdot C - b_{n-1}\cdot 2^n\cdot C \end{aligned}$$

In other words, we can compute $B\times C$ by performing an unsigned multiplication on B and C followed by a subtraction of n-bit left-shifted C if B is negative.

Extending this to 2-way SIMD multiplication is straightforward. We can perform two n-bit signed-unsigned multiplications in a SIMD fashion by performing one $(3n+1)\times n$-bit unsigned multiplication, followed by at most two subtractions.

However we can reduce the number of subtractions to one by performing a *signed* multiplication for the $(3n+1)\times n$-bit multiplication. In this case the upper $2n$-bit (i.e., $A\times C$)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

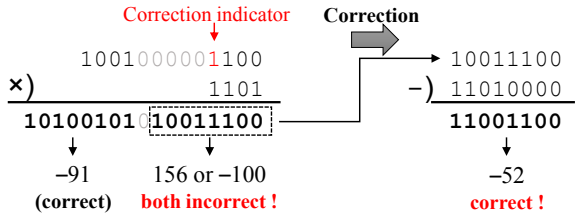


Fig. 2: Example SIMD execution: -7×13 and -4×13.

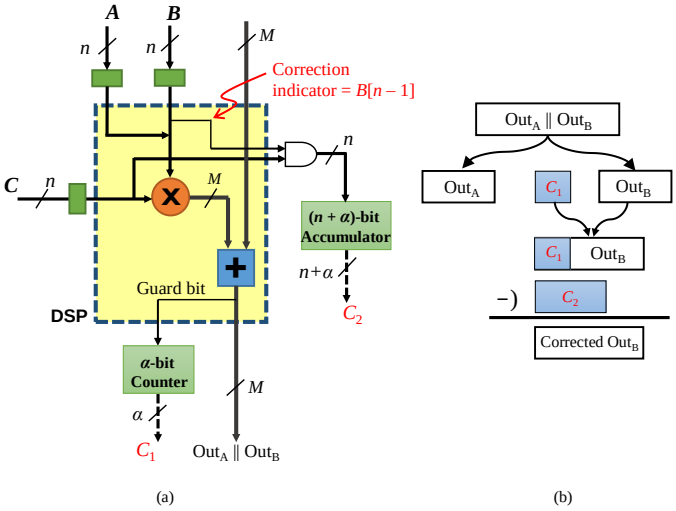


Fig. 3: (a) Datapath of our Double MAC architecture, where $n = 8$ and $M = 48$, and (b) how it works.

is already correct (there is no need for correction), and only the lower $2n$-bit needs a correction if B is negative, as demonstrated by the example in Fig. 2.

The correction term could be added in-place if we do just one multiply-add operation. For accumulation of a large number of multiplication results, however, we must delay doing the corrections until all accumulation is done because we have only one guard bit.[1] Instead the correction terms are accumulated separately in a small accumulator, whose value is later subtracted from the main accumulator. We determine the size of extra accumulators in the next section.

C. Accumulation and Our Double MAC Architecture

Fig. 3 illustrates our Double MAC architecture. One DSP block can implement both a 2-way SIMD multiplier and a 2-way SIMD adder. Both the multiplier and the adder have a correction output signal, which is shown as thin arrows in the figure. The correction outputs are accumulated into a separate register or counter to be used later for final adjustment.

In the final adjustment two terms need to be added/subtracted. The adder-correction term C_1, which comes from the carry-out counter, has no overlap with the main accumulator output, and thus can be just concatenated. Therefore we need only one addition—the subtraction of the multiply-correction term C_2 from the concatenation result.

[1]Even if there are enough guard bits, the subtractor cannot be created inside the DSP block, which precludes the possibility of in-place correction as a DSP block has three operands only.

The width α of the carry-out counter is determined from the number of values accumulated, V, as follows: $\alpha \geq \log_2 V$. In the case of our baseline CNN accelerator (see Section II-A), $V = (K^2N/T_N)$, thus α should be no less than $\log_2(K^2N/T_N)$ for every layer of the CNN, where K is the size of convolution filter in one dimension. Similarly the width of the accumulator for multiplier correction terms is $n + \alpha$.

The post-accumulation adjustment can be done easily by using an extra adder of $(n + \alpha)$-bit width that resides outside the DSP block. There is no runtime overhead due to this adjustment, since the adjustment is done simultaneously while new values are loaded into the MAC.

IV. CNN Accelerator Based on Our Double MAC

Our Double MAC is specialized in two ways: it shares one operand and the shared operand is unsigned. We show how to overcome these limitations as well as the precision issue in the context of CNN accelerators.

A. Sufficiency of Shared-Operand SIMD Multiplier

As explained in Section II-A, our baseline accelerator has T_M outputs, performing at any given time one step of a convolution between T_N inputs and weight matrices. More precisely, at any given cycle the MAC array in Fig. 4(a) updates the (r, c)-element of T_M output feature maps for a certain iteration index (i, j) as follows.

$$Y_m[r,c] \leftarrow Y_m[r,c] + \sum_n^{T_N} W_{m,n}[i,j] \cdot X_n[r+i, c+j] \quad (1)$$

This is repeated K^2 times while the index (i, j) iterates over the size of the convolution filter, $K \times K$. Omitting the indexes we have a simpler form:

$$y_m \leftarrow y_m + \sum_n^{T_N} w_{m,n} \cdot x_n, \quad (2)$$

where y_m, $w_{m,n}$, and x_n are each an element of their respective matrices.

Comparing the computations of two output feature maps at the same image location, $Y_m[r, c]$ and $Y_{m+1}[r, c]$, we see that only the weight parameters are different but the input values are the same.

$$Y_{m+1}[r,c] \leftarrow Y_{m+1}[r,c] + \sum_n^{T_N} W_{m+1,n}[i,j] \cdot X_n[r+i, c+j] \quad (3)$$

This leads to our Double MAC array sharing input values as follows, where "|" represents concatenation with an appropriate number of zeros inbetween:

$$y_m|y_{m+1} \leftarrow y_m|y_{m+1} + \sum_n^{T_N} w_{m,n}|w_{m+1,n} \cdot x_n \quad (4)$$

The Double MAC array has the same number of input ports, T_N, but half the number of output ports, $T_M/2$.

Another way to exploit our Double MAC array in a convolution layer is to pair neighboring features in the same

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

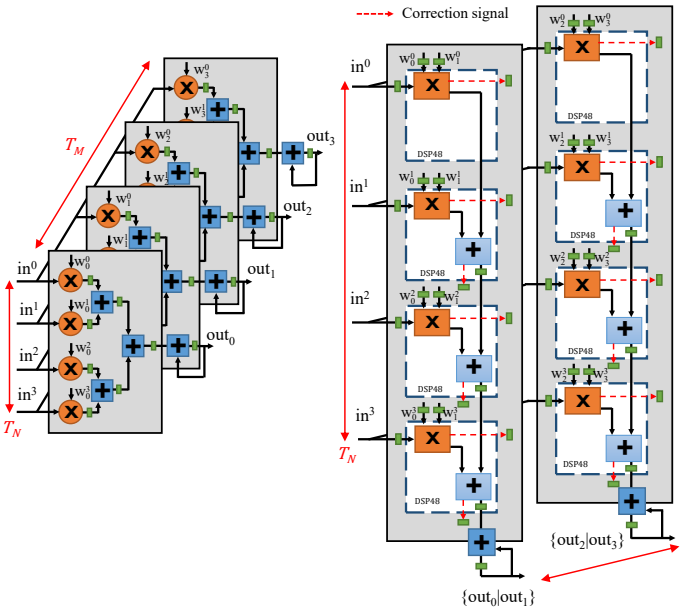


Fig. 4: Computation engine of a CNN accelerator. (a) A MAC array from [2] based on floating-point MACs and (b) our modified version supporting our Double MACs.

output feature map. This works if the CNN accelerator applies the MAC array along the R-direction (i.e., the output row direction), such as those parallelizing along the M-R loops [5]. Suppose our accelerator parallelizes along the M-R loops. Then the MAC array would update T_R elements of T_M output feature maps for a certain input feature map index n, a certain output column index c, and a certain iteration index (i, j) as follows.

$$Y_m[r, c] \leftarrow Y_m[r, c] + W_{m,n}[i, j] \cdot X_n[r + i, c + j] \quad (5)$$

Again this is repeated K^2 times while the index (i, j) iterates over the size of the convolution filter. But unlike in the previous case, this takes in only one input feature map, X_n, and thus needs to be repeated N times, the number of input feature maps. Omitting the indexes we can write

$$y_m[r] \leftarrow y_m[r] + w_m \cdot x[r, c] \quad (6)$$

Then our Double MAC array computes the following using shared weight parameters.

$$y_m[r]|y_m[r+1] \leftarrow y_m[r]|y_m[r+1]+w_m \cdot x[r, c]|x[r+1, c] \quad (7)$$

Similarly, the Double MAC array can be applied to accelerators parallelizing along the M-C loops [5] and along the M-R-C loops [14]. In the remainder of this paper we assume that the accelerator parallelizes along the M-N loops like [2], [4].

B. Computation Engine: Double-MAC Array

Fig. 4 illustrates the architecture of our Double-MAC array, contrasting it with that of the original MAC array in [2]. In the original MAC array 5 DSP blocks are used to implement a pair of multiplier and adder, as they use 32-bit floating-point precision. We maximize the compute density by using 16-bit and 8-bit fixed-point precision, which can be implemented with just 1 DSP block, with one caveat: in the 1 DSP-per-MAC case, about half of the adders in the adder tree cannot share a DSP block with a multiplier. This is because a DSP block can take up to three operands, and therefore cannot support $A \times B$ and $C + D$ unless they are chained. The unmatched adders need DSP blocks of their own, significantly decreasing resource utilization, or require many LUTs to implement.

Instead our MAC array uses a pipelined adder cascade. This way, adders can always be matched with multipliers. Further since we are not using adder tree, our MAC array can support any T_N values that may not be 2^k. We use this enhanced version as the baseline for our evaluation in Section V.

As mentioned, the SIMD multipliers in our MAC array perform signed-unsigned multiplications, with input feature maps having unsigned values. This naturally fits with CNNs with ReLU (Rectified Linear Unit) activation layers, which produce non-negative outputs, to be used as input for the following layer. Next we discuss how to convert input/output feature map values to unsigned numbers.

C. Conversion into Unipolar-Input CNN

An arbitrary convolutional layer can be converted into an equivalent *unipolar-input CNN*, which is one that has unsigned, nor non-negative, values in all its input feature maps. The conversion is surprisingly simple, and can be done off-line after training by adjusting bias values. There is very little runtime overhead.

The computation of a convolutional layer, including bias b_m, can be represented as $Y_m + b_m J = \sum_n^N W_{m,n} * X_n + b_m J$, where J is a matrix of ones. Suppose that the input feature map X_n is a matrix of k-bit signed fixed-point numbers, whose range is $[-2^{k-1}, 2^{k-1} - 1]$. We define an unsigned input feature map X'_n as follows.

$$X'_n = X_n + 2^{k-1} J \quad (8)$$

Then we can find the new bias values b'_m from the requirement that the new outputs generated from using the unsigned input feature maps must equal the original outputs.

$$\begin{aligned} Y_m + b_m J &= \sum_n^N W_{m,n} * X'_n + b'_m J \\ &= \sum_n^N W_{m,n} * (X_n + 2^{k-1} J) + b'_m J \\ &= Y_m + \sum_n^N W_{m,n} * 2^{k-1} J + b'_m J \end{aligned}$$

Thus $b'_m = b_m - 2^{k-1} \sum_n^N \sum_{i,j} W_{m,n}[i, j]$.

This bias adjustment can be done off-line and therefore incurs no runtime overhead. The only overhead at runtime is the conversion of signed input into unsigned for (8), which is simply to flip the MSB of X_n (and interpreting it as unsigned value) and can be done using one invertor.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

D. Improving Accuracy with Shift-only Scaling Scheme

One important concern for a reduced-precision scheme such as ours is that it may result in lower accuracy in the application output. In this section we analyze the precision impact of our Double MAC architecture and present a very low-cost scheme to mitigate the impact.

Recall that all three operands (i.e., two multiplicands and one common multiplier) of a Double MAC have 8-bit precision but the output has $(16 + \alpha)$-bit (lower word) and 31-bit $(= 48 - 17$, upper word) precision. These output precisions should be enough for DNN applications. Post-MAC steps such as max-pooling, normalization, and activation function are independent of our Double MAC and therefore outside of our consideration. Also, unlike approximate addition/multiplication approaches [24], our Double MAC architecture is exact for the given operands, and there is no accuracy loss inside. Thus the only loss of accuracy by our architecture is caused by the truncation (or rounding) of input operands, which come from input feature maps (including the primary input for the first layer) and weight parameters.

In other words, while the correct output, Y, is given by $Y_i = \sum W_{ij}X_j + B_i$ (where B is the bias value), our Double MAC architecture can only compute $Y'_i = \sum[W_{ij}][X_j]+[B_i]$, where $[\cdot]$ represents an 8-bit quantization operation whether by truncation or rounding.

Thus to mitigate the loss of accuracy due to quantization we use scaling. In particular we pre-scale both input feature maps and weight parameters (including bias), and later scale the output back after all accumulation is finished. The scaling factor can be set differently for different layers. Let s_x and s_w be the scaling factors for input feature maps and weight parameters, respectively, of a certain layer. Then the scaled versions are obtained as follows.

$$X' = s_x X \quad \text{and} \quad W' = s_w W \tag{9}$$

By scaling B (bias) and MAC output by $s_x s_w$ and $1/(s_x s_w)$, respectively, we can bring the MAC output back its original range.

$$Y'_i = \left(\sum [s_w W_{ij}][s_x X_j] + [s_x s_w B_i]\right) / (s_x s_w) \tag{10}$$

$$= \sum [s_w W_{ij}]/s_w \cdot [s_x X_j]/s_x + [s_x s_w B_i]/(s_x s_w) \tag{11}$$

From (11) we can see that scaling can help reduce quantization error by shifting the input range. But scaling may incur runtime overhead due to the multiplication of X by s_x in (9) and that of the accumulation result by $1/(s_x s_w)$ in (10). Therefore, to get multiplier-free scaling we must use a power of 2 value for both s_x and s_w. Weight parameters and bias values can be scaled off-line, which incurs no runtime cost.

Parameters s_x and s_w are per-layer parameters, whose values are determined through profiling. We determine these parameters such that $P \simeq 0.99$, where P is the probability of scaled values being within the range of 8-bit precision, which may be [-128, 127] for signed values and [0, 255] for unsigned values.[2] From profiling we know the exact distribution of the input data and therefore can find the best scaling parameters. Or one can simply use the first-order statistics assuming that the input data follow a normal distribution. Then the scaling parameter can be calculated as $128/(\mu+3\sigma)$ for signed data or $256/(\mu+3\sigma)$ for unsigned data, where μ and σ are the mean and standard deviation, respectively. We round the scaling parameter to the nearest power of 2.

[2]One may use different ranges than shown here, but what is important is to use the same ranges for *different layers*.

TABLE I: Resource requirements of different data precisions

Data type	DSP	LUT	FF
Floating-point 32-bit [2]	5	349	355
Fixed-point 32-bit [2]	4	0	0
Fixed-point 16-bit	1	0	0
Fixed-point 8-bit	1	0	0
Fixed-point 8-bit, Double MAC	0.5	11	12

V. Experiments

A. Experimental Setup

We apply our technique to the convolution layers of two real-life CNNs: AlexNet [9] and VGG [10]. AlexNet consists of 8 layers, including 5 convolution layers. There are several configurations of VGG, in this paper we use the configuration that has 16 layers, including 13 convolution layers. Both networks use the ReLU activation function. Our baseline CNN accelerator is from [2] with an adder cascade enhancement as described in Section IV. The accelerator performs MAC operations only, which accounts for the majority of computation.

We have extended the Caffe framework [27] to obtain testbench data for RTL validation as well as to measure the output quality of limited-precision networks. To simulate the fixed-point behavior of the convolutional layer we have added a quantization layer before every convolutional layer. The output of convolutional layers is used to generate test vectors. The configuration for n-bit fixed-point is $(n, 0)$, which means all n bits are dedicated for the integer part (including sign bit) and none is assigned for the fractional part, as we find that this configuration gives the best recognition performance. Our accuracy analysis is done with n ranging from 4 to 11.

For our performance comparison we set the clock frequency to 280MHz, which agrees with our synthesis results, and the memory bandwidth to 9 GB/s based on our RTL simulation.

B. Synthesis Results

Table I compares the resource requirements of one MAC unit for different precisions. The data for floating point and 32-bit fixed-point are from [2], shown here for comparison.

We have implemented our Double MAC array in Verilog RTL and validated its functionality using Vivado simulator. The MAC array is synthesized using Vivado 2015.2 targeting the Xilinx Virtex7 485T FPGA, which has 2,800 DSP blocks. The MAC array has two key parameters, T_M and T_N, also called *tile parameters*, that impact the throughput of the accelerator. Their optimal values, shown in Table II, are found using exhaustive search for AlexNet. As expected, with the same level of DSP utilization our SIMD architecture can implement a MAC array twice large than the non-SIMD version.

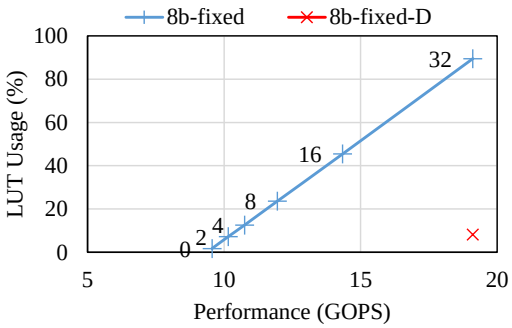
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

The table also reports the area and maximum frequency of the synthesized circuit. Though our Double MAC array consumes more LUT and FF, which is due to the correction circuit, the additional resource usage is not high. On the other hand, being able to turn the extra resources into performance improvement can be an advantage, especially when the resources are typically underutilized in CNN accelerators unless a floating point data type is used (see the table).

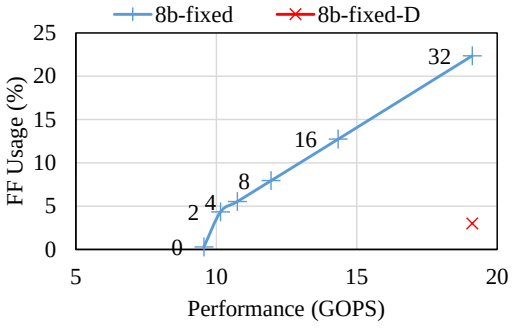
C. Comparison Against Using LUTs

One downside of our Double MAC is a higher usage in terms of LUTs and FFs (Flip-Flops). Particularly since one can synthesize additionals MACs using the extra LUTs and FFs, comparing it with our scheme is very interesting. We compare 8-bit fixed case with and without our Double MAC scheme. For the baseline case we synthesize additional MACs using LUTs so that we can compare the two cases under the iso-resource condition. In particular, we use the array size of (T_M, T_N) = (64, 64) for the Double MAC case. For the baseline case, the subarray of size (T_M, T_N) = (32, 64) is implemented using DSP blocks, but extra rows of MACs along the T_M direction are implemented using LUTs. In all cases the DSP utilization is exactly the same.

Fig. 5 shows the results. For the 8-bit fixed case, the number of extra rows is varied from 0, 2, 4, 8, 16, and 32. One can see from the graphs that there is an almost linear relationship between the number of extra rows of MACs and the additional LUT/FF required. Thus in order to realize 2× speedup in the 8-bit fixed case, one needs to utilize about 89% LUT and 22% FF. On the other hand, our scheme uses only a very small amount of additional resource that is similar to that of 2 extra rows for the 8-bit fixed case, i.e., (T_M, T_N) = (32+2, 64). Put in another way, using the same level of additional LUT/FF resources, ours can generate 2× speedup, or 100% higher performance without any adverse effect, whereas naïvely using LUTs and FFs can generate only about 6% (= 32 vs 34 rows) higher performance, which clearly demonstrates the advantage of our scheme.



(a) LUT usage



(b) FF usage

Fig. 5: Comparison against using LUTs for additional MACs. In the case of 8b-Fix, the number on the left side of the marks represents the number of extra rows of MACs.

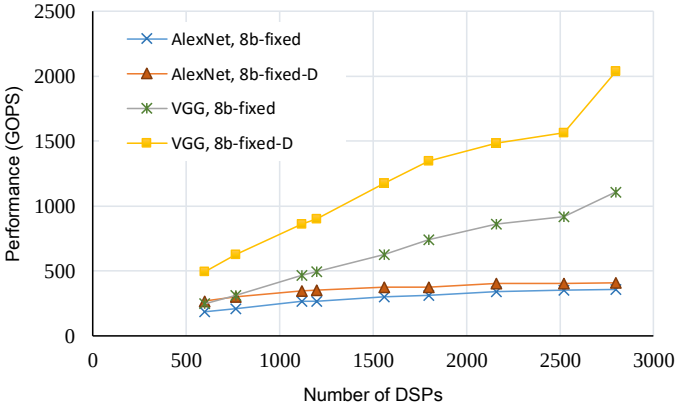


Fig. 6: Performance vs. Number of DSPs.

D. Performance Scalability

Fig. 6 shows how the performance of two CNNs (AlexNet and VGG) on the two CNN accelerators (8-bit fixed-point baseline vs. our Double-MAC array) varies as the number of DSPs is changed. Other conditions such as memory bandwidth are kept unchanged. The number-of-DSPs value is taken from the DSP counts of existing Xilinx FPGA devices [33].

We first observe that different CNNs have very different performance scaling patterns. VGG's performance on the baseline array scales almost linearly as the number of DSPs increases whereas AlexNet's performance is nearly saturated at around 400 GOPS. Consequently when our Double MAC is applied, the performance improvement by our scheme is drastically different between the two applications. While our Double MAC achieves significant speedup on VGG (84%), its performance with AlexNet is not as impressive (14%). Nonetheless, we observe a common pattern that our Double MAC array can achieve the performance level of the baseline at twice the number of DSPs. In other words our Double MAC array can successfully double the *effective* number of DSPs for both CNNs, even when judged by the computation rate improvement.

E. Detailed Performance Analysis

To understand the difference between the two CNNs we analyze the runtime at the layer level. Table III and Table IV compare the run time of each convolutional layer of AlexNet and VGG, respectively. We also show the ratio of 16-bit and 8-bit fixed-point implementation over our method in terms of runtime, power and energy. Both applications run on Virtex7 485T FPGA (the DSP count is 2,800 but limited to 80% usage limit). The tile parameters suggest that our Double MAC array

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II: Synthesis results for optimal tile parameters

	AlexNet				VGG		
	32b-float [2]	16b-Fix	8b-Fix	8b-Fix-D	16b-Fix	8b-Fix	8b-Fix-D
Optimal (T_M, T_N)	(7, 64)	(11, 192)	(32, 64)	(64, 64)	(64, 35)	(35, 64)	(64, 64)
Max frequency (MHz)	280	280	280	280	280	280	280
DSP usage (%) (*)	80	75	73	73	80	80	73
LUT usage (%)	61.30	0.39	1.12	16.98	2.08	1.23	16.98
FF usage (%)	33.87	0.11	0.32	8.88	0.56	0.35	8.88
BRAM (KB)	4608	656	343	369	1295	1248	1632

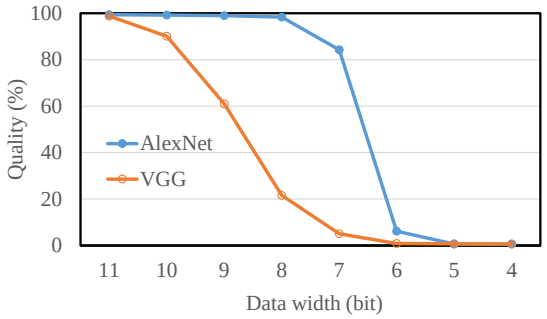
*Note: DSP usage is limited to 80% as in the previous work [2].

TABLE III: AlexNet performance comparison (unit: ms, W, J)

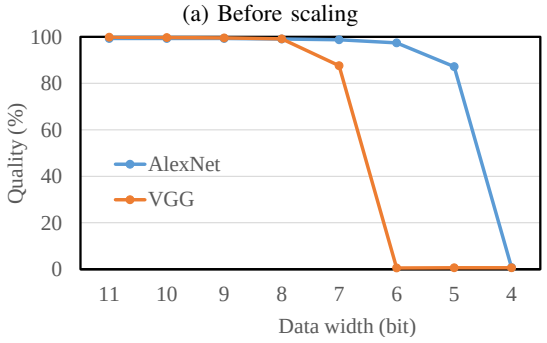
Layer	GOP	16b-Fix	8b-Fix	8b-Fix-D	Ratio	
L1	0.21	1.31	1.31	1.31	1.00	1.00
L2	0.45	0.33	0.26	0.13	2.50	2.00
L3	0.30	0.13	0.13	0.08	1.54	1.54
L4	0.22	0.10	0.10	0.06	1.54	1.54
L5	0.15	0.10	0.07	0.04	2.23	1.49
Total	1.33	1.96	1.86	1.63	1.20	1.14
Power		4.81	4.35	5.78	0.83	0.75
Energy		9.43	8.10	9.42	1.00	0.86
(T_M, T_N)		(11, 192)	(32, 64)	(64, 64)		

TABLE IV: VGG performance comparison (unit: ms, W, J)

Layer	GOP	16b-Fix	8b-Fix	8b-Fix-D	Ratio	
L1	0.17	3.23	1.61	1.61	2.00	1.00
L2	3.70	3.23	3.23	1.61	2.00	2.00
L3	1.85	1.61	1.61	0.81	2.00	2.00
L4	3.70	3.23	3.23	1.61	2.00	2.00
L5	1.85	1.61	1.61	0.81	2.00	2.00
L6	3.70	3.23	3.23	1.61	2.00	2.00
L7	3.70	3.23	3.23	1.61	2.00	2.00
L8	7.40	1.51	1.61	0.81	1.88	2.00
L9	3.70	3.02	3.02	1.61	1.88	1.88
L10	3.70	3.02	3.02	1.61	1.88	1.88
L11	3.70	0.93	0.76	0.46	2.05	1.66
L12	3.70	0.93	0.76	0.46	2.05	1.66
L13	0.92	0.93	0.76	0.46	2.05	1.66
Total	41.79	29.71	27.67	15.07	1.97	1.84
Power		5.93	5.81	7.85	0.75	0.74
Energy		176.16	160.79	118.39	1.49	1.36
(T_M, T_N)		(64, 35)	(35, 64)	(64, 64)		



(a) Before scaling



(b) After scaling

Fig. 7: Relative quality with various bit-widths of AlexNet and VGG, with and without scaling. Quantization is applied to convolution layers only.

has exactly twice the number of MAC units than that of the baseline. Thus it is possible to achieve 2× speedup, which is what we see in many layers.

However in the first layer, the speedup over the 8-bit fixed-point implementation is zero with either application. This is because in those layers only some MAC units are actually utilized due to smaller layer size compared with tile size. In the AlexNet, for instance, the first layer has layer parameters $M = 64$ and $N = 3$, but the tile parameters found are $T_M = 64$ and $T_N = 64$, which means that, for N, 61 out of 64 MACs are not utilized during the first layer. Because of this, the first layer receives no benefit from the increased MAC array size of our Double MAC. To make matters worse L1 dominates the AlexNet's runtime, dragging down the overall speedup. VGG has the same issue in the first layer (having $M = 64, N = 3$) but the first layer is not as dominant.

On the other hand, the first layer's performance can be changed by the choice of unroll dimensions (tile dimensions) when designing the MAC array. If one chooses, for instance, R and C loops [34] instead of M and N, the first layer will scale much better. In fact R and C have the highest values in the first layer, so doubling tile parameters may directly translate into increased performance.

Another point is the speedup varies layer by layer. This is because the tile parameters are sub-optimal for some layers, causing either under-utilization or fragmentation. Such erratic behavior also explains why we see a nonlinear curve in the VGG-SIMD case in Fig. 6.

We also estimate FPGA power dissipation using Xilinx Power Estimator [35] with numbers obtained from our synthesis report. Since our method uses extra resources to implement the correction circuit, our power dissipation is higher than that of the baseline. In case of AlexNet, the runtime improvement achieved by our method is not enough to compensate for higher power, resulting in higher energy consumption. However, in case of VGG, since our Double MAC array executes significantly faster than the baseline, it achieves about 33% energy reduction compared to the 8-bit fixed-point implementation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2824280, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

F. Accuracy and Effect of Scaling

Fig. 7 shows the output quality (i.e., our top-5 accuracy divided by that of the floating-point implementation) of our quantized version as we vary the precision from 4-bit through 11-bit, with and without our scaling optimization. We use the entire validation set of ILSVRC-2012.

For this experiment we run the entire network on Caffe, and fixed-point quantization is applied only in front of convolution layers (for input feature map and weight parameter data). Scale values are all powers of 2, so that free scaling-back is possible.

Our results reveal that despite their large sizes, both CNNs are surprisingly resilient up to a certain precision, beyond which the quality starts to quickly deteriorate. Our scaling optimization can delay this quality degradation by at least 1 or 2 bits as shown in the graphs. Also it is worth noting that the quality degradation at 8-bit quantization, which is the precision used in our SIMD evaluation, is quite low (less than 1% when quantization is applied to convolution layers only), which agrees with earlier results (e.g., [7]) that show very low accuracy degradation up to 8-bit quantization on AlexNet and VGG.

VI. Conclusion

We presented how to increase the computation rate of CNN accelerators on FPGAs by packing multiple MAC operations into one DSP blocks of off-the-shelf FPGAs. By exploiting the context in which MAC operations are used, our method can strike a good balance between usability and implementation overhead. We have validated our proposed architecture through Verilog simulation and FPGA synthesis, and evaluated it using a state-of-the-art CNN accelerator, which shows that our Double MAC can increase computation throughput of a CNN layer often by twice, and achieve signficant performance improvements on the network level ranging from 14% to more than 80% over an already optimized accelerator solution depending on the hyper-parameters of the CNN and the FPGA size. All these are done without sacrificing the output quality significantly. This improvement in performance can directly translate into energy saving, which can make accelerator solutions even more appealing as compared to GP-GPU solutions.

References

[1] D. Nguyen, D. Kim, and J. Lee, "Double MAC: Doubling the performance of convolutional neural networks on modern FPGAs," in *Design, Automation and Test in Europe (DATE '17)*, Mar. 2017.

[2] C. Zhang *et al.*, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 161–170.

[3] S. Cadambi *et al.*, "A programmable parallel accelerator for learning and classification," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 273–284.

[4] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 26–35.

[5] M. Peemen *et al.*, "Memory-centric accelerator design for convolutional neural networks," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, Oct 2013, pp. 13–19.

[6] *Nvidia Tesla P100*, Nvidia, available at http://www.nvidia.com/object/tesla-p100.html (last accessed May 4, 2017).

[7] N. Suda *et al.*, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 16–25.

[8] "7 series DSP48E1 slice user guide," Xilinx, User Guide UG479, Sep. 2016.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks." in *NIPS*, P. L. Bartlett *et al.*, Eds., 2012, pp. 1106–1114.

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

[11] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 609–622.

[12] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, pp. 262–263.

[13] K. Bong *et al.*, "14.6 a 0.62mw ultra-low-power convolutional-neural-network face-recognition processor and a CIS integrated with always-on haar-like face detector," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017, pp. 248–249.

[14] A. Rahman, J. Lee, and K. Choi, "Efficient fpga acceleration of convolutional neural networks using logical-3d compute array," in *Design, Automation and Test in Europe (DATE '16)*, Mar. 2016.

[15] S. Han *et al.*, "Eie: Efficient inference engine on compressed deep neural network," *SIGARCH Comput. Archit. News*, vol. 44, pp. 243–254, Jun. 2016.

[16] A. Rahman *et al.*, "Design space exploration of FPGA accelerators for convolutional neural networks," in *Design, Automation and Test in Europe (DATE '17)*, Mar. 2017, pp. 1147–1152.

[17] S. Gupta *et al.*, "Deep learning with limited numerical precision," *CoRR*, vol. abs/1502.02551, 2015.

[18] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, pp. 668–673, 2014.

[19] W. Wen *et al.*, "A new learning method for inference accuracy, core occupation, and performance co-optimization on truenorth chip," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 18:1–18:6.

[20] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *CoRR*, vol. abs/1511.00363, 2015.

[21] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," *CoRR*, vol. abs/1511.06393, 2015.

[22] S. Venkataramani *et al.*, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *Low Power Electronics and Design (ISLPED), 2014 IEEE/ACM International Symposium on*, Aug 2014, pp. 27–32.

[23] Q. Zhang *et al.*, "Approxann: An approximate computing framework for artificial neural network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, March 2015, pp. 701–706.

[24] S. S. Sarwar *et al.*, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, pp. 145–150.

[25] K. Kim *et al.*, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *53rd Annual ACM/IEEE Design Automation Conference (DAC '16)*, Jun. 2016.

[26] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *54th Annual ACM/IEEE Design Automation Conference (DAC '17)*, Jun. 2017, pp. 29:1–29:6.

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

[28] J. Cho, H. Chang, and W. Sung, "An FPGA based SIMD processor with a vector memory unit," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, May 2006.

[29] B. Mahmood and M. Al Jbaar, "Design and implementation of simd vector processor on fpga," in *Innovation in Information Communication Technology (ISIICT), 2011 Fourth International Symposium on*, Nov 2011, pp. 124–130.
[30] P. Bonnot *et al.*, "Definition and SIMD implementation of a multi-processing architecture approach on FPGA," in *Design, Automation and Test in Europe, 2008. DATE '08*, March 2008, pp. 610–615.
[31] S. Perri *et al.*, "A high-performance fully reconfigurable FPGA-based 2-D convolution processor," in *Microprocess. Microsyst., vol. 29*, 2005, pp. 381–391.
[32] S. Grys, "Signed multiplication technique by means of unsigned multiply instruction," *Comput. Electr. Eng.*, vol. 37, pp. 1212–1221, Nov. 2011.
[33] "7 series FPGAs data sheet: Overview," Xilinx, Data Sheet DS180, Mar. 2017.
[34] C. Farabet *et al.*, "Cnp: An fpga-based processor for convolutional networks," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, Aug 2009, pp. 32–37.
[35] "Xilinx power estimator user guide," Xilinx Inc., User Guide UG440 (v2015.3), Sep. 2015.

Sugil Lee received the B.S. degree in mathematical science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2017. He is currently a master's student in department of computer science and engineering in Ulsan National Institute of Science and Technology (UNIST), Ulsan, Korea. His research interests include deep neural networks, natural language processing, and automatic speech recognition.

Daewoo Kim received the B.S. degree in computer science from University of Ulsan, Ulsan, Korea, in 2015. He is currently a master's student in department of computer science and engineering in Ulsan National Institute of Science and Technology (UNIST), Ulsan, Korea. His research interests include processor design, accelerator design for emerging application such as deep learning, and high level synthesis.

Dong Nguyen received the M.S. degree at Ulsan National Institute of Science and Technology (UNIST), Ulsan, South Korea in 2016. His research interests include parallel programming and deep learning on accelerator platform such as GPU and FPGA. Currently he is a senior software engineer at Robert Bosch Engineering Vietnam and is responsible for developing embedded software for automotive control units.

Jongeun Lee (M'01) received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in electrical engineering and computer science all from Seoul National University, Seoul, Korea, in 1997, 1999, and 2004, respectively. Since 2009 he has been on the faculty of UNIST, Korea, in the school of electrical and computer engineering. His research interests include architectures and compiler for reconfigurable architectures, and deep learning acceleration.