IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 1, 2019, accepted November 18, 2019, date of publication November 29, 2019,
date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2956841

A Novel Software-Defined Convolutional Neural Networks Accelerator

YUFENG LI[1] **AND YANKANG DU**[2]
[1]Shanghai University, Shanghai 200444, China
[2]National Digital Switching System Engineering Technology Research Center, Zhengzhou, China

Corresponding author: Yankang Du (duyankang@163.com)

ABSTRACT The convolutional neural network (CNN) has been widely applied in computer vision applications. Due to the intensive computation, the general central processing unit (CPU) processors are not efficient to meet the real-time requirement. Various hardware accelerators based on the application specific integrated circuit (ASIC) and field programmable gate array (FPGA) have been designed to accelerate CNN models during the inference phase. The data flow architecture has been extensively adopted because of the high parallelism. However, given the continual development in the computer vision field, CNN models have become increasingly diverse. Thus, CNN accelerators based on data flow architectures face an emerging challenge to maintain high throughput while coping with various CNN models. In this paper, we design a software-defined architecture to solve this. The goal of this study is to make the hardware change as the application changes to achieve high flexibility and high performance. In our proposed architecture, all the parts can be software-defined to cope with different CNN models. A flexible software-defined processing element (PE) array is designed to compute different weight filter sizes. In addition, a software-defined data reuse technique based on two ideal reuse cases is proposed to ensure that all the parameters need to be loaded only once during the computing phase. To support this reuse technique, we also propose the software-defined on-chip buffer so that the weight and image buffers share one dynamic buffer. By using the sparsity property of the input feature map, the full-connected (FC) layer is accelerated. About 88% of the FC weight parameters can be skipped when loading the VGG-16 model. Finally, we implemented this software-defined accelerator on the FPGA. Compared to the other FPGA based accelerators, our proposed accelerator can preserve high performance while maintaining flexibility.

INDEX TERMS Software-defined, data flow, accelerator, convolutional neural networks.

I. INTRODUCTION

In recent years, the convolutional neural network (CNN) has emerged as the prevalent model for the machine learning and computer vision. Now, deep CNNs are widely used in a broad range of real-life applications, such as image classification, object detection and image segmentation [1]–[4]. Due to the intensive computation and huge external data access for the CNN algorithm, general central processing unit (CPU) processors are unable to meet real-time demands. Thus, specialized accelerators should be designed. The CNN algorithm can be accelerated during both the training and inference phases. Here, we focus on the inference phase, which is widely used in embedded vision system.

The associate editor coordinating the review of this manuscript and approving it for publication was Naveed Akhtar.

It is well known that the massive multiply and accumulate operations (MAC) and the enormous numbers of parameters are the two main bottlenecks faced by CNN hardware acceleration methods. CNN accelerators based on the application and specific integrated circuits (ASIC) [5]–[13] and field-programmable gate array (FPGA) [14]–[21] have been designed to solve these problems. Basically, two types of architecture exist: instruction flow and data flow. Recently, data flow architectures have garnered greater attention due to their higher parallelism capabilities [6]–[13], [17]–[19]. For instances, Eyeriss [6] is a high energy-efficient data flow architecture. It proposed the row stationary (RS) data flow, which can exploit data reuse of the input feature map and weight filters. This technique can help minimize the energy consumption caused by the on-chip data movement. Tensor processing unit (TPU) [7] uses systolic matrix multipliers to accelerate the convolutional operations. On average, the TPU

can achieve a speed of approximately 15X-30X compared to its contemporary GPU or CPU. This provides a broad prospect for data flow architectures. Work on data flow architectures is on-going and dominates in the current research work. In addition to the usual parallelism acceleration, model compression [22] and low precision quantitation [23] provide different ideas to accelerate CNN algorithms. Here, we do not involve these techniques and focus only on the common parallelism acceleration architecture.

With the continuous development in the compute vision field, CNN models have become diverse [25]–[28]. Apart from the two basic bottlenecks mentioned above, the current accelerators based on data flow also faces the challenge of how to match the various CNN models while maintaining high performance. For the TPU, despite its high-performance, the multiplier utilization is low when the network size is lower than the systolic matrix size [7]. Many works [10]–[13] used reconfiguration techniques to enhance the hardware flexibility. Venieris and Bouganis proposed a synchronous dataflow architecture based on hardware reconfiguration techniques [19]. This method can cope with various CNN layers with the mostly suitable architecture and hardware resources by reconfiguring the FPGA. This idea is quite attractive, and it can achieve speedups of up to 2.94× compared with the other FPGA-based architectures. However, the hardware must be reconfigured for different sub-graphs. The acceleration performance would be constrained by the reconfiguration time.

Overall, the diversity of CNN models poses a substantial challenge to current CNN accelerators based on the data flow. We expected that the accelerator can achieve high throughput while keeping certain flexible. In other words, the hardware can be altered according to the needs of different applications to provide the mostly suitable hardware architecture and resources. Similar to the reconfiguration architecture, we propose a software-defined architecture that can maintain high throughput when dealing with various CNN models. The software-defined architecture considers both the software and hardware aspects to design the accelerator. The main contribution of this paper is as follows.

We designed a software-defined architecture for CNN accelerators. Compared to previous research works, all the hardware modules in our architecture can be reconfigured according to the requirement of different CNN models. A software-defined data reuse technique is used to reduce the data communication between the on-chip buffer and double data rate (DDR). In addition, we develop a full tool chain to translate CNN models described in Caffe into hardware reconfiguration instructions. Experimental results on the FPGA show that our proposed accelerator can maintain high throughput when coping with various CNN models.

The organization of the rest paper is organized as follows: Section II introduces our proposed software-defined architecture. Section III proposes two optimized techniques using the sparse property of the CNN models. Section IV implemented this CNN accelerator based on the Zynq platform and give a comparison with other FPGA based accelerators. Section V gives a conclusion.

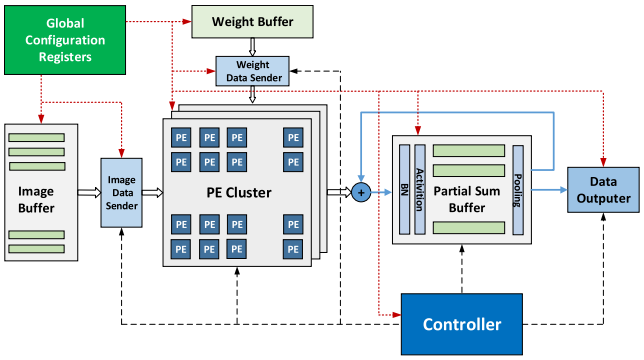


FIGURE 1. Software-defined architecture for the CNN accelerator.

II. SOFTWARE-DEFINED ARCHITECTURE

A. ARCHITECTURE REVIEW

Fig. 1 shows our proposed software-defined architecture based on data flow. The biggest difference from the other data flow architecture is that all the parts, even the data computing modes, can be reconfigured. The global configuration registers module is designed to reconfigure the other hardware parts. In our architecture, the image buffer and weight buffer are directly linked to the processing element (PE) cluster. The computing results of PE cluster are streamed into the partial sum buffer.

To reduce the amount of data communication between the DDR and the on-chip buffer, the batch normalization (BN), activation and pooling operations are streamed and conducted in the partial sum buffer. The partial sum buffer is composed of many static random-access memory (SRAM) banks, allowing it to receive computed results from the PE cluster in parallel.

B. SOFTWARE-DEFINED CONVOLUTIONAL PE ARRAY

The processing element (PE) architecture which determines flexibility and throughput plays a key role in CNN accelerators. Many PE architectures, such as the spatial 2D-PE array [6] and the systolic matrix [7], have been designed to perform convolutional operations. The TPU is not highly efficient when dealing with small layer size, while the spatial 2D-PE array can perform well when applied to different convolutional sizes with more complex design. In all, the design of a PE array is a trade-off between complexity, throughput and flexibility.

From the view of simplicity and flexibility, we designed the PE structure as shown in Fig. 2. Different from the other normal PE design that contains only one multiplier [6], [8], we included nine multipliers in our PE because the 3×3 and 1×1 weight filters dominate is in the current CNN models. Thus, we gave primary consideration to these two filter sizes.

In addition to the multipliers, this PE contains nine weight registers and nine image registers. The image registers are organized as shift register modes as shown in Fig. 2.

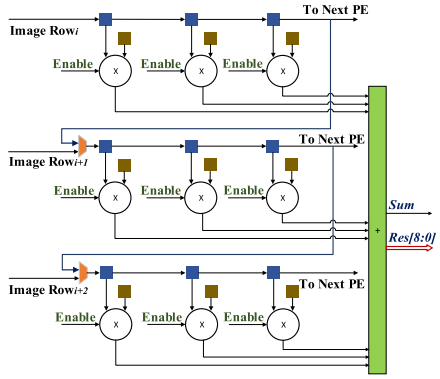


FIGURE 2. Software-defined PE structure containing nine multipliers.

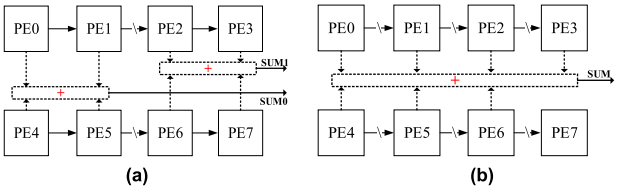


FIGURE 3. Hardware configure for the 5 × 5 and 7 × 7 weight filter, (a) 5 × 5 weight filter, (b) 7 × 7 weight filter.

The communication between PE is based on these image shift registers. Each multiplier has an enable signal to determine whether this multiplier works. For the multiplier that does not work, its output is set to zero.

This PE contains two kinds of outputs. One is the sum of the 9 multiple results, the other is the 9 separate multiple results. In this way, the PE can conduct one 3 × 3 convolutional operation or nine 1 × 1 convolutional operation. And based on the inner connection of the image shift registers, this PE can also do one row convolutional operation for a 7 × 7 weight filter.

Based on our proposed PE structure, both 5 × 5 and 7 × 7 weight filters can be calculated. Fig. 3 shows a model where eight PEs are adopted. This PE array can deal with two 5 × 5 convolutional operations and one 7 × 7 convolutional operation. When 16 PEs are used, one 11 × 11 convolutional operation could be calculated.

In Eyeriss, the rows of the input feature map are reused across PEs to reduce the on-chip data movement [6]. This method greatly reduces the on-chip energy cost. The stride and padding are two important parameters when computing the convolution operations. The stride is the number of pixels with which the weight filter slides, horizontally or vertically. Zero Padding occurs when we add a border of pixels all with value zero around the edges of the input images. we can also specify whether or not to use padding according to the input feature map size and the weight filter size.

For our design, we adopted a similar row-reuse idea. Fig. 4 illustrates one instance. The example assumes a weight filter size of 3 × 3 and a stride of 1. The feature map data are stored on the on-chip buffer in a two-dimensional mode. Fig. 4 shows the input for each PE in the array. It can be

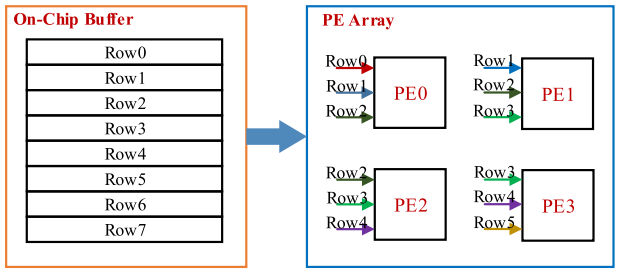


FIGURE 4. Row data reused example.

TABLE 1. Parallelism parameter for one PE array to process one convolutional layer.

Symbol	Description
N_i	Number of input feature map
N_o	Number of output feature map
W_o	Width of the output feature map
H_o	Height of the output feature map
N_p	Number of input feature map one PE array concurrently computes
N_r	Number of rows one PE array concurrently produces for each computed input feature map
N_c	Number of weight filter channels one PE array concurrently computes

got that the row data are extensively reused. Row1 is reused twice, and Row2 and Row3 are reused three times. In addition to reusing the image rows, the row-reuse technique greatly reduces the number of SRAM bank, which can mitigate the design complexity. As shown in Fig. 4, only 6 banks are needed when using four PEs.

Our accelerator uses two types of parallelism: the inter - output parallelism and inter-kernel parallelism [29] as defined below:

(a) Inter-output Parallelism: Different output feature maps are totally independent of each other, and theoretically can all be computed in parallel. In reality, the parallelism is limited due to the limited resources.

(b) Inter-kernel Parallelism: Each image pixel in each output feature map is the result of a set of convolutions. It is possible to compute all of them concurrently and then sum them to get the final results.

These PEs are organized in 2D-array mode in the PE array. The PE array can be configured flexibly. Table 1 presents the parallelism parameter for one PE array. The theoretical parallelism degree (Tpd) for one PE array to accelerate one convolutional layer can be expressed in forum (1).

$$\mathrm{Tpd} = N_p^* N_r^* N_c \tag{1}$$

As the image data is arranged in two-dimensional row mode. The number of on-chip buffer banks has a close relationship with the parameters shown in Table 1.

The PE cluster contains several PE arrays. Additionally, a PE cluster can be configured to address either inter-output parallelism or inter-kernel parallelism.

TABLE 2. Peak multiplier utilization for different weight sizes in the pe array.

Filter Size	Peak Utilization
1x1	100%
3x3	100%
5x5	69.4%
7x7	77.8%
11x11	84.0%

Table 2 presents the peak multiplier utilization for different weight sizes in the PE array. Expect for the 3×3 and 1×1 weight filter sizes, a certain level of multiplier waste exists. However, because the 5×5, 7×7 and 11×11 weight filters are less frequently used, we thought that this waste was tolerable. For instance, the ResNet [24] model adopts 7×7 weight filter size only in the first layer and all the other layers use filter sizes of 1×1 and 3×3.

In addition to the convolutional layer, there also exists the full-connected (FC) layer. The FC layer can be treated as a special convolutional layer in which the size of the input feature map is 1×1, the size of weight filter is 1×1, the padding is 0 and the stride is 1. In our architecture, the FC layer would be computed as the convolutional layer is when the weight parameter size of the FC layer is not dominant. When the weight parameter size in the FC layer is dominant, using this approach would result in an extended communication time. We would use a special technique to cope with this problem described below.

C. SOFTWARE-DEFINED DATA REUSE TECHNIQUES

Due to the limited on-chip memory resources, the image and weight parameters cannot be all loaded on-chip. Usually, the data reuse techniques should be adopted to solve this problem.

As known, moving data from external memory to the on-chip buffer consumes large amounts of energy. Related studies have revealed that the energy consumed for one data movement operation from the DDR is orders of magnitude larger than the that for a multiply operation [6]. Image division and channel division reuse modes are widely used in CNN accelerators [8]. Shin *et al.* proposed a mixed division reuse technique to further reduce off-chip access [8]. Although the mixed division is efficient, it might result in the parameters being reloaded from the external memory to the on-chip buffer several times.

For a CNN accelerator, the data reuse mode plays an important role because it determines the control and data computing flows. For a given convolutional layer, we find that two ideal data reuse modes exist, as shown in Fig. 5. One ideal reuse mode is that the input feature map data are stored on-chip, the other is that the weight parameters for one layer are all stored on-chip. We termed the reuse modes in Fig. 5 (a) and Fig. 5(b) as the ideal reuse mode 1 and ideal reuse mode 2, respectively.

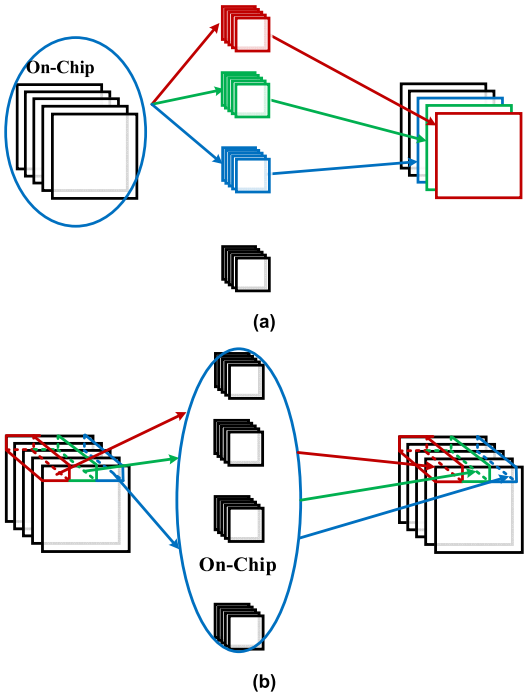


FIGURE 5. Two optimized data reuse techniques, (a) all image data in one layer is stored on-chip, (b) all weight data in one layer is stored on-chip.

The data computing flows for these two ideal data reuse techniques are different. For the ideal reuse mode 1, only the weight parameters for one channel need to be loaded to compute the corresponding output feature map. This process can continue until all the channel weight parameters are loaded. In contrast, for the ideal reuse mode 2, only one tile of the input map needs to be loaded to compute the corresponding tile of the output feature map. After all the tiles of the input feature map have been computed, the entire output feature map can be obtained.

Because all the weight parameter and input image feature map data need be loaded only once, these two ideal reuse modes are quite attractive. Thus, the question is: can these two ideal data reuse modes be implemented? When the on-chip buffer is sufficiently large, both modes are possible. However, in most real-world cases, the on-chip buffer is limited. In our work, we found that the sizes of the input feature map and weight parameter vary among different convolutional layers, as presented in Fig. 6. Usually, the feature map size is larger in the shallow layers. As the convolutional layer becomes deeper, the weight parameter size becomes dominant.

Based on this property, we proposed the reconfiguration reuse techniques by using ideal reuse mode 2 for the shallower layer and applying the ideal reuse mode 1 for the deep layers. This approach greatly reduces the amount of the off-chip access.

In real environments, when neither the input feature map nor the weight parameter can be loaded, these two ideal reuse models cannot be used. In this case, we use a split approach. For instance, when adopting ideal reuse mode 2, the input

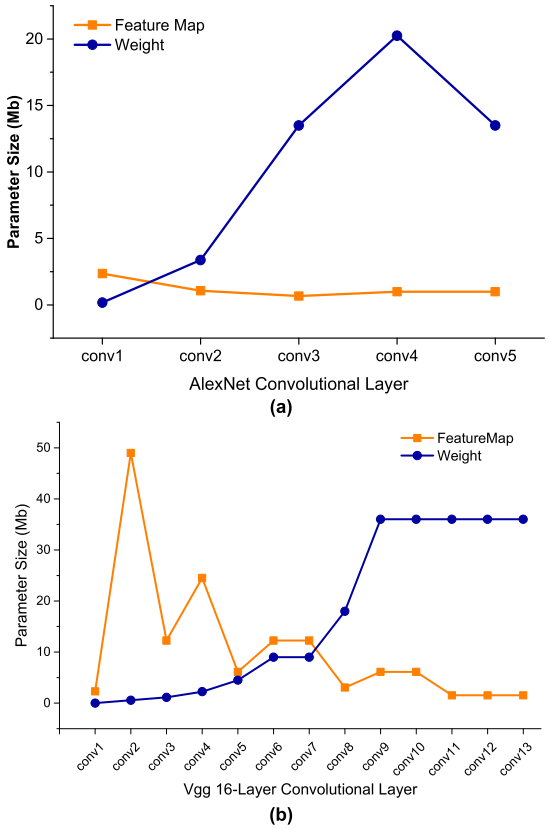


FIGURE 6. Weight parameter and feature map data size in different convolutional layers, (a) AlexNet model, (b) VGG-16 model.

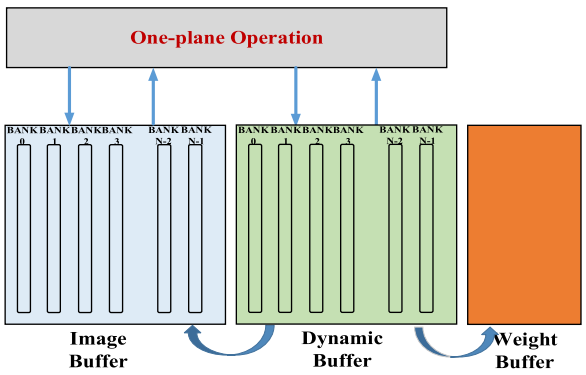


FIGURE 7. Origination of on-chip buffer.

feature map is split into several tiles to be loaded on-chip. This causes the weight parameters to be reloaded several times.

D. SOFTWARE-DEFINED ON-CHIP BUFFER

To support the software-defined PE array and the data reuse techniques, we designed the software-defined on-chip buffer. In contrast to traditional accelerator that include only the weight and image buffers, we propose the dynamic buffer presented in Fig. 7. Depending on the reuse mode demand, the dynamic buffer can be allocated to either the image or the weight buffer. For instance, for the ideal reuse mode 1,

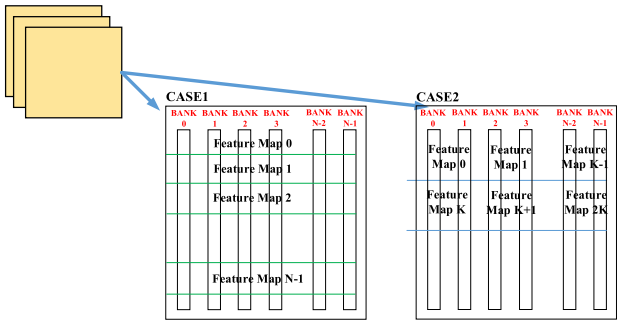


FIGURE 8. Image Data arrangement on the on-chip buffer.

the input feature map would be loaded on-chip as much as possible. The dynamic buffer would be allocated to the image buffer. This approach allows us to exploit the on-chip buffer resources.

In addition to the dynamic buffer, we implemented a one-plane operation inside the on-chip buffer. The one-plane operation includes the Eltwise [24], Pooling (2×2 size, 3×3 size), Avg Pooling [25], [26] and other operations. By doing so, we can provide a complete end-to-end, flexible CNN acceleration that is not limited solely to the convolutional acceleration.

In our design, the input image data is stored on the on-chip buffer using the two-dimensional mode. The data arrangement on the on-chip buffer can also be software-defined. Fig. 8 present two cases. In other words, in our on-chip buffer, the number of banks used to store one feature map is reconfigurable.

Overall, the data arrangement for the on-chip buffer should be set according to the PE array requirements.

III. OPTIMIZATION TECHNIQUE USING THE SPARSE PROSPERITY

The above discussion provided a detailed explanation of our proposed software-defined CNN accelerator. In this section, we would introduce an optimization technique that utilizes the sparseness prosperity of the input feature map.

In CNN models, due to the introduction of the ReLU active function [24]–[27], the input feature maps for each layer are actually quite sparse. Fig. 9 presents the sparseness of each layer during the CNN inference phase. We can see that the sparseness is particularly prominent in the deeper layers.

Here, we capitalize on the sparseness property of the input feature map to optimize our accelerator. Both convolutional and the FC layers are considered.

For the convolutional layer, we followed the same approach used in [6], [8] that the multiplication operation is skipped when the input image data in the PE is "0". Using this approach, the number of the multiplication operations can be greatly reduced, which reduces the on-chip energy cost.

For the FC layers, the weight parameters in the FC layer are distributed as shown in Fig. 10. For the VGG-16 layer, the weight parameter proportion occupies 89.9%, which results

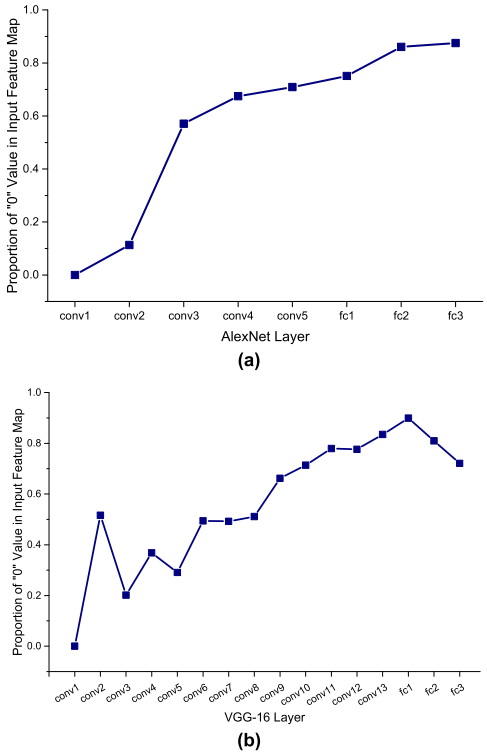


FIGURE 9. Proportion of "0" values in the input feature map of each layer, (a) AlexNet model, (b) VGG-16 model.

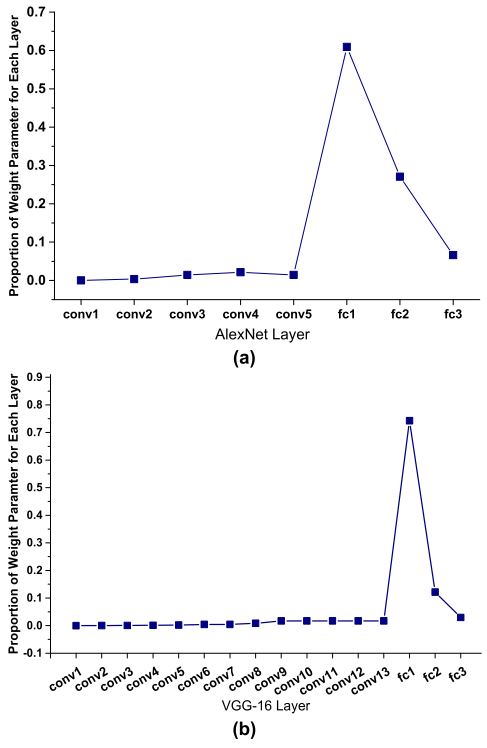


FIGURE 10. Weight parameter proportion of each layer, (a) AlexNet model, (b) VGG-16 model.

in long data load time. Usually, the FC layers are pruned to compress the size of the weight parameters [16]. Current CNN models such as the ResNet and MobileNet adopted average pooling to reduce the number of weight parameters in the FC layer.

For these CNN models where the weight size of the FC layer is dominant, treating the FC layer the same as a convolutional layer would involve a large data load time. To solve this problem, we exploited the sparseness property of the FC layer.

As shown in Fig. 9, the number of "0" value in the input of FC layer is also large. We capitalize on this property to optimize the FC layer. The typical formula for the FC layer is shown in Equation (2).

$$\begin{aligned}&[I_0, I_1, \cdots\cdots, I_{n-1}]\\ &\quad\times\begin{bmatrix} W_{0,0} & W_{0,1} & \cdots\cdots & W_{0,m-1}\\ W_{1,0} & W_{1,1} & \cdots\cdots & W_{1,m-1}\\ W_{2,0} & W_{2,1} & \cdots\cdots & W_{2,m-1}\\ \vdots & & & \\ W_{n-1,0} & W_{n-1,1} & \cdots\cdots & W_{n-1,m-1}\end{bmatrix}\\ &= [R_0, R_1, \cdots\cdots, R_{m-1}]\end{aligned} \tag{2}$$

Two kinds of computing approaches exist, as shown in Equations (3) and (4).

$$R_i = [I_0, I_1, \cdots\cdots, I_{n-1}]\begin{bmatrix} W_{0,i}\\ W_{1,i}\\ \vdots\\ W_{n-1,i}\end{bmatrix} \tag{3}$$

$$\begin{aligned}[R_0, \ldots, R_{m-1}] &= I_0{}^*[W_{0,0}\cdots W_{0,m-1}]\\ &\quad+\cdots+I_{n-1}{}^*[W_{n-1,0}\cdots W_{n-1,m-1}]\end{aligned} \tag{4}$$

For Equation (3), one column of the weight is loaded to compute the Ri. For Equation (4), one row of the weight is loaded to be multiplied by Ii.

As shown in Fig. 9, the number of "0" values in the matrix $[I_0\ I_1\ I_2\ \ldots.\ I_{n-1}]$ is large. Based to this property, we adopted Equation (4) to compute the FC layer. When the value of I_i is "0", the corresponding row in the weight $[W_{i,0}\ W_{i,1}W_{i,2}\ \ldots.\ I_{i,n-1}]$ does not need to be loaded. Thus, only a small proportion of the weight parameter in the FC layer needs to be loaded. This approach can largely mitigate the volume of data movements from the external memory to the on-chip buffer.

IV. EVALUATION

A. EXPERIMENTAL SETUP AND IMPLEMENTATION

To verify our proposed software-defined CNN accelerator, we implemented the proposed architecture on the Xilinx Zynq ZC706 platform. The Zynq architecture includes two major parts: the programming logic (PL) and the processing system (PS). The CNN accelerator is mainly implemented in the PL part; the PS part is used to control the accelerator. Fig. 11 shows our experimental platform. A notebook is connected to the ZC706 development board.

Table 3 shows the details of the parameters for our proposed CNN accelerator. According to the resources available on the Zynq Z-7045, we implemented 4 PE arrays and set

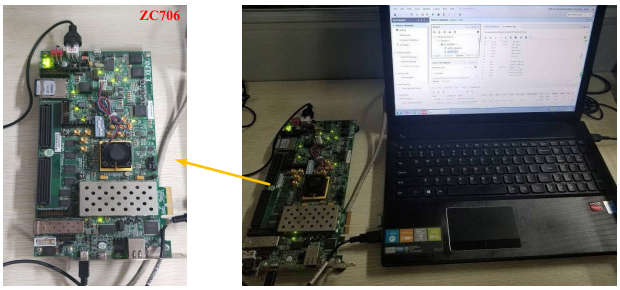


FIGURE 11. Experimental platform.

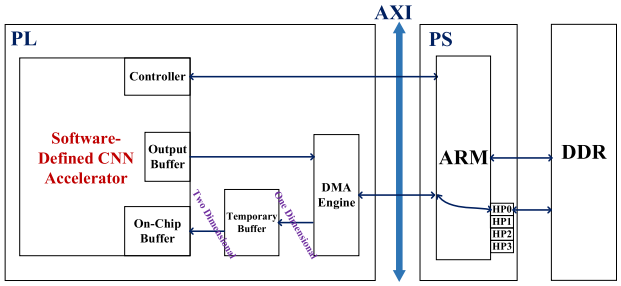


FIGURE 12. System implementation overview.

TABLE 3. Implementation Details of our CNN Accelerator on the Xinlix Zynq Z-7045.

Parameter	Value
PE Arrays	**4**
PEs in each PE array	**16**
On-chip image buffer banks for each PE array	**18**
On-chip dynamic buffer banks for each PE array	**18**
Partial sum buffer banks	**16**

TABLE 4. Computing parameter details for one PE array.

Filter Size	Stride	Parallel processing for one PE Array			SRAM Banks
		Feature Map Number	Channel Number	Image Rows	
1x1	1	8	8	2	**16**
3x3	1	1	1	18	**18**
		1	2	10	**10**
	2	1	2	17	**17**
5x5	1	1	1	8	**8**
	2	1	1	11	**11**
7x7	2	1	1	9	**9**
	3	1	1	10	**10**
11x11	-	1	1	11	**11**

to inter-kernel parallelism. Each PE array contains 16 PE arranged in the 4×4 mode. To simplify the design, the input image row data can be moved only in the horizontal PEs based on the image shift registers described in Section II. There is no connection in the vertical direction. The PEs configuration to cope with the 5×5 and 7×7 weight filter size is the same as that shown in Figure 4.

There are 18 on-chip image buffer banks for each PE array is 18 and each bank is set to a width of 64 bits. There are 16 partial sum buffer banks that is a partial sum buffer can concurrently receive 16 data values from the PE cluster.

The parameters detailed for one PE array are shown in Table 4. The parallel processing parameters for a PE array are closely related to the on-chip buffer banks. From the table, we can see that the maximum number of required banks is 18.

Fig. 12 shows an overview of the implemented system. Data exchange between the PL and the PS is based on an AXI bus. The PS part sends the address of the configuration instruction in the DDR to the accelerator and then sends a start signal. After the CNN accelerator receives the start signal, it reads the configuration instructions from the DDR using the DMA engine. Then, the CNN accelerator operates according to the instructions.

TABLE 5. FPGA resource utilization.

Resource	BRAM	DSP	LUT	FF
Available	545	900	218600	437200
Used	537	592	134034	116223
Utilization	98.53%	65.78%	61.31%	26.58%

Because the input feature map data in the DDR are stored in the one-dimensional model, one temporary buffer is used to change the one-dimensional data into the two-dimensional on-chip data. To overlap the data transfer time with the computing time, the on-chip weight buffer, the on-chip image buffer, and the partial sum buffer are all equipped with double-buffers that operate in ping-pong mode.

Regarding data precision, we use 16-bit fixed point precision as have most previous research works [16]–[19]. The decimal points can be dynamically adjusted based on the ranges of pixel values in different layers. Table 5 shows the FPGA resource utilization for our implementation.

We design a series of configure registers, including configurations for the PE array, controller, data sender, data loader, global buffer configuration and so on, for our software-defined CNN accelerator. By configuring these registers, the hardware behavior can change to accommodate various CNN layers. A configuration instruction is 32 bits. Each instruction contains sufficient information to configure one or more registers. To configure one CNN layer only approximately one hundred configuration instructions. Compared to the computing time, the configuration time can be ignored.

We implement a full tool chain to translate the CNN models described in Caffe into hardware configuration instructions as shown in Fig. 13. During the translation process, the CNN compiler intelligently judges which hardware reconfiguration is most suitable for each CNN layer. The generated hardware configuration instructions, the pre-processed weight parameters and the image data are stored in the DDR. After the PS send the start signal, the accelerator in the PL

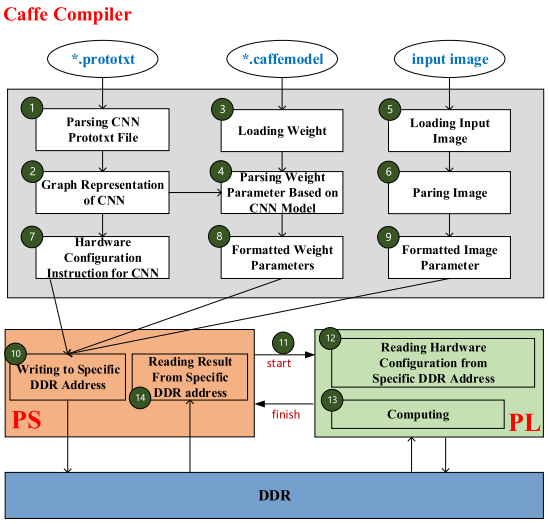


FIGURE 13. Full tool chain for our proposed CNN accelerator.

reads the instruction from the DDR and proceeds as indicated by the instruction. When PL acceleration is complete, it sends a finished signal to the PS. Then, the PS reads the CNN results from a specific DDR address.

B. COMPARISONS WITH PREVIOUS WORKS

Because we use only the FPGA to implement the software-defined CNN accelerators, we compare our work only with existing FPGA-based accelerators. Many FPGA-based accelerators have been designed for different FPGA platforms. It is hard to give an absolutely fair comparison between different research works. Because our research goal is to provide a unified software-defined architecture that can cope with different CNN models rather than to optimize a CNN accelerator for one target FPGA platform, we adopt the performance density metric, which yields a relatively fair comparison across different FPGA platforms.

As is known, the number of multipliers and the clock frequency are directly related to the throughput. The value of performance density (PD) is expressed as shown in Equation (5).

$$\text{PD} = \text{Throughput}/(\text{Clock} * \text{Multiplier Number}) \quad (5)$$

The multiplier number in Equation (5) usually equals the number of used DSP. In some special cases, one DSP can be treated as two multipliers.

The CNN compiler is used to translate the CNN models descripted in Caffe into the hardware configure instructions. By executing these configure instructions, the CNN accelerator can run an image. The throughout is obtained by Equation (6). The run time is just referred to the infer time for an image.

$$\text{Throughput} = \text{Total Operation Number/Run time} \quad (6)$$

Tables 6 and 7 respectively shows performance comparisons for VGG-16 and ResNet models using different

TABLE 6. Performance comparison of different accelerators wit VGG-16.

		Qiu[16]	Meloni [18]	Venieris[19]	This Work
Platform		Xinlix Zynq Z-7045	Xinlix Zynq Z-7045	Xinlix Zynq Z-7045	Xinlix Zynq Z-7045
Clock (MHz)		150	140	125	140
Precision		16bit fixed	16-bit fixed	16-bit fixed	16-bit fixed
DSP		780	864	855	592
Throughput (GOP/s)	Conv	187.89	169.7	155.81	130.41
	Full	136.97	122.58	-	109.18
Throughput Density	Conv	1.61x10-3	1.40x10-3	1.45x10-3	1.57x10-3
	Full	1.17x10-3	1.01x10-3	-	1.31x10-3
Flexibility		Low	High	Low	High

TABLE 7. Performance comparison of different accelerators with ResNet.

	Meloni [18]	Ma [17]	Venieris[19]	This Work	
Platform	Xinlix Zynq Z-7045	Arria-10 GX 1150	Xinlix Zynq Z-7045	Xinlix Zynq Z-7045	
Clock (MHz)	140	200	125	140	
Precision	16-bit fixed	16 bit fixed	16-bit fixed	16-bit fixed	
DSP	864	1518	896	592	
Model	ResNet-18	ResNet-152	ResNet-152	ResNet-18	ResNet-152
Throughput (GOP/s)	58	710.30	188.18	94.28	115.56
Throughput Density	$4.6\text{x}10^{-4}$	$1.17\text{x}10^{-3}$	$1.68\text{x}10^{-3}$	$1.03\text{x}10^{-3}$	$1.39\text{x}10^{-3}$
Flexibility	High	High	Low	High	

FPGA-based accelerators. In [17], a DSP can be used as two 16×16 multipliers. For ResNet, only the convolutional layers are considered.

Flexibility in this work is measured based on whether an accelerator can work with different CNN models without re-designing the code and re-downloading the bit stream.

The works in [16] and [19] have the lowest flexibility because their accelerators are suitable for only limited CNN models. Thus, when the models change, the code must be re-designed.

Due to the way our accelerator capitalizes on "0" value utilization in the FC layer, it achieves the best performance density for the VGG-16 model. When considering only convolutional layers, the performance density of our architecture is only slightly lower than that of the research work in [16].

Meloni *et al.* proposed an end-to-end reconfiguration architecture that can deal with different CNN models [18]. This is the same as our work does. However, compared to their approach, our accelerator is better, especially for ResNet-18.

Venieris and Bouganis proposed a synchronous dataflow method to accelerate CNN models [19], in which the CNN model is divided into several sub-graphs. They designed a specific architecture for different sub-graphs and generated a corresponding bit stream file. This approach provides the most suitable architecture and hardware resources by re-downloading the bit stream. Now, this approach achieves the best performance. As this approach uses the dynamic partial reconfiguration in the FPGA, now it can be only applied to the FPGA architecture. In addition, the bit stream file re-download time might become a bottleneck when dealing with different sub-graphs.

For the ResNet-152 model, the throughput density of our accelerator is well below that of the accelerator in [19]. This gap occurs mainly because our accelerator and the work in [17] do not consider the BN and Eltwise operations when computing the throughput. However, those operations were considered in [19]. After unifying the total operations, the throughput of work [19] (not including the BN and Eltwise operation) is 1.29×10^{-3}. As shown in Table 4, the performance of our designed accelerator is better than the one proposed in [19].

Zeng *et al.* proposed an FFT-based acceleration technique and achieved high throughput [20]. However, we think this framework is not truly end-to-end because the pooling layer, FC layer and convolutional operations for the 11x11 filter are executed on execute on the CPU. Additionally, this FFT-based approach cannot effectively accelerate the 1x1 weight filter.

Overall, our proposed software-defined accelerator maintains high performance while preserving flexibility. The time needed to reconfigure each layer requires only approximately 100 clock cycle. This has a great advantage over the partial dynamic configuration time used in [19].

V. CONLUSION

In this paper, we designed a CNN accelerator based on the software-defined architecture. All the parts of our architecture can be software-defined, allowing it to provide the most suitable hardware structure to cope with various CNN models. To mitigate the amount of off-chip data access, we proposed the software-defined data reuse techniques. In addition, the software-defined on-chip buffer is designed to support these data reuse techniques. The experimental results indicate that the intermediate results and weight parameters can mostly be loaded only once. By using the sparseness property of the input feature map, we propose a computation technique for FC layers. About 88% of the FC weight parameters can be skipped during loading for the VGG-16 model. Finally, we implemented this software-defined accelerator on the FPGA platform. Compared to the other FPGA-based accelerators, our proposed accelerator maintains high performance while preserving flexibility.

Software-defined hardware approach is becoming increasingly popular. The idea is quite suitable to accelerate the continuously varied CNN models. In the future, we plan to demonstrate our proposed software-defined architecture on an ASIC. In addition, we are involved in efforts to give our complier support for more CNN frameworks, such as the TensorFlow, PyTorch.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. F. Li, "Large-scale video classification with convolutional neural networks," in *Proc. IEEE Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2014, pp. 1725–1732.

[3] O. Russakovsky, J. Deng, H. Su, and J. Krause, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[4] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.

[5] 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 *Proc. 19th Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2014, pp. 269–284.

[6] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.

[7] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, and R. Boyle, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[8] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 240–241.

[9] M. T. Hailesellasie and S. R. Hasan, "MulNet: A flexible CNN processor with higher resource utilization efficiency for constrained devices," *IEEE Access*, vol. 7, pp. 47509–47524, Apr. 2019.

[10] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[11] W. Choi, K. Choi, and J. Park, "Low cost convolutional neural network accelerator based on bi-directional filtering and bit-width reduction," *IEEE Access*, vol. 6, pp. 14734–14746, Mar. 2018.

[12] L. Du, Y. Du, Y. Li, J. Su, Y.-C. Kuan, C.-C. Liu, and M.-C. F. Chang, "A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 198–208, Aug. 2017.

[13] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Mar. 2018, pp. 461–475.

[14] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2015, pp. 161–170.

[15] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-S. Seo, and Y. Cao, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, Feb. 2016, pp. 16–25.

[16] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, and Y. Wang, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2016, pp. 26–35.

[17] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks," in *Proc. IEEE Int. Conf. Field-Program. Logic Appl. (FPL)*, Sep. 2017, pp. 1–8.

[18] P. Meloni, A. Capotondi, G. Deriu, M. Brian, F. Conti, D. Rossi, L. Raffo, and L. Benini, "NEURA ghe: Exploiting CPU-FPGA synergies for efficient and flexible CNN inference acceleration on Zynq SoCs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 1, pp. 1–22, Dec. 2017.

[19] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 326–342, Feb. 2019.

[20] H. Zeng, R. Chen, C. Zhang, and V. Prasanna, "A framework for generating high throughput CNN implementations on FPGAs," in *Proc. ACM/SIGDA Int. Symp. (FPGA)*, Feb. 2018, pp. 117–126.

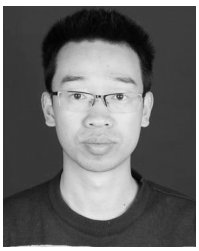
[21] N. Shah, P. Chaudhari, and K. Varghese, "Runtime programmable and memory bandwidth optimized FPGA-based coprocessor for deep convolutional neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 5922–5934, Apr. 2018.

[22] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, Jun. 2016, pp. 243–254.

[23] F. Conti, P. D. Schiavone, and L. Benini, "XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2940–2951, Nov. 2018.
[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[25] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861
[27] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 6848–6856.
[28] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 7132–7141.
[29] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," in *Proc. IEEE 21st Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2016, pp. 575–580.

YUFENG LI received the B.E. and Ph.D. degrees from the National Digital Switching System Engineering Technology Research Center (NDSC), Zhengzhou, China, in 1998 and 2008, respectively.

He was an Associate Professor with the NDSC, from 2013 to 2018. He is currently a Professor with Shanghai University. His research interests include broadband information networks and network security technology.



YANKANG DU received the B.E. degree from Xi'an Jiaotong University, Xi'an, China, in 2009, and the M.S. and Ph.D. degrees from the National University of Defense Technology, Changsha, China, in 2011 and 2015, respectively.

He is currently an Associate Professor with the National Digital Switching System Engineering Technology Research Center (NDSC). His main research interests include very large integrated circuits design and AI hardware acceleration.