IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 8, 2017, accepted April 26, 2017, date of publication April 28, 2017, date of current version June 7, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2699229

Enhanced Efficiency 3D Convolution Based on Optimal FPGA Accelerator

HAI WANG, (Member, IEEE), MENGJUN SHAO, YAN LIU, AND WEI ZHAO
Xidian University, Xi'an 710071, China

Corresponding author: Hai Wang (wanghai@mail.xidian.edu.cn)

This work was supported by the Open Foundation of the State Key Laboratory for Manufacturing Systems Engineering under Grant sklms201511.

ABSTRACT This paper presents an enhanced efficiency 3-D convolution operator based on optimal field programmable gate array (FPGA) accelerator platform. The proposed system takes advantages of the intermediate data delay lines, implemented in an FPGA, to avoid loading repetition of the input feature maps. This 3-D convolution accelerator performs 268.07 giga operations per second at 100-MHz operation frequency, with 330-mW power consumption. We experimentally demonstrate the enhanced efficiency of the proposed convolution accelerator, in comparison with the conventional technologies. The proposed 3-D convolution accelerator may find interesting applications in neural networks and video processing.

INDEX TERMS Accelerator architectures, neural networks, convolution, field programmable gate arrays.

I. INTRODUCTION

Over the past few decades, the Convolutional Neural Networks (CNN), as an advanced machine learning algorithm, have found numerous applications, including computer vision and speech recognition [1]–[3]. Despite growing use of the two-dimensional convolutional neural networks, they still suffer from inefficiency of CPUs, showing up during the implementation procedure [4]. To overcome this issue, various accelerator technologies have been recently proposed, including Graphics Processing Unit (GPU), Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA).

GPU is one of the best platforms for accelerating the convolutional neural networks, especially while the training process is involved [3], [5], [6]. A single GPU presents an outstanding performance, but its high power consumption obstructs its application in embedded systems. Moreover, GPUs are restricted by the inflexible parallel calculation units, and require reforming adaptable calculation models. Another technology, ASIC chip, is well-known for its superior performance and high energy efficiency [7]–[9], while it suffers from less flexibility and high developing cost.

FPGA is a promising CNN acceleration platform which shows superior performance over the aforementioned technologies. Recently, FPGA-based accelerators have spurred huge attention since they provide high performance, good energy efficiency, short development cycle as well as the reconfigurable characteristics. Previous works on the FPGA-based 2D CNN accelerators have shown increasing throughputs [4], [10]–[12]. Peeman et *al*. focused on maximization of the reuse of on-chip data [10]. Concentrating on the optimization of both resources and communication bandwidth, Zhang et *al*. presented an implementation that achieved a peak performance of 61.62 Giga Floating-point Operations Per Second (GFLOPS) under 100MHz operation frequency on a Virtex-7 FPGA [4]. In [11], the authors leveraged all sources of parallelism in CNNs and their work ran 84.2 GFLOPS on a Virtex-7 FPGA. Ovtcharov et *al*. reported a specialized hardware based on the Xeon and FPGA co-processing for data centers to reach the speed of 233 images per second with Catapult Server and Arria 10 GX1150 FPGA [12]. As for 2D convolution accelerators, in [13] the authors presented a MultiWindow Partial Buffering (MWPB) scheme for 2D convolution operators and achieved a speed of one pixel per clock. Carlo et *al*. dramatically decreased the area required by MWPB scheme for a better integration with embedded systems [14]. Work in [15] reduced the look up table resource usage in systolic array architecture. References [16] and [17] separated the 2D convolution to reduce the computational complexity. Besides, a speed of one pixel per clock is achieved in [16], and a speed of 194 frames per second (f/s) is achieved in [17]. However, FPGAs accelerate pre-trained 2D CNN models and the convolution operations may occupy over ninety percent of the computation time due to the convolution calculation burden [18].

Exploiting the aforementioned acceleration approaches, the 2D CNNs have been widely developed. Meanwhile, the 3D CNN calculation module has been emerged and found various fascinating applications, including targets tracking and human action recognition [19]. The 3D convolution takes advantage of one-time capturing, where more information of multiple contiguous frames are obtained in one shot. However, this represents a greater computational burden, especially for the embedded real-time video processing. As a result, the main challenge of such a system is to reach an acceptable performance and power consumption. The 3D convolution based on FPGA accelerator may stand as an alternative technique to overcome these issues, however, few studies have been reported in this area.

In this paper, a 3D convolution operator based on optimal FPGA accelerator is presented. The optimal operation is achieved by taking advantages of the Intermediate Data Delay Lines (IDDL) to avoid pixels loading repetition. Moreover, the proposed system exhibits high calculation performance and low power consumption due to the tailored hardware of the 3D convolution. Specifically, we make the following contributions. We propose a 3D convolution accelerator design with IDDLs and tailored hardware. As a case study, we implement a 3D convolution accelerator that achieves a performance of 268.07 GOPS and 2.65× that of the fastest commercially available GPU.

The paper is organized as follows. Section II describes the operation principle and properties of the proposed 3D convolution. Section III presents the optimization and implementation of the FPGA accelerator. The experimental demonstration of the proposed system will be given in Section IV. Finally, Section V concludes the paper.

II. OPERATION PRINCIPLE

Figures 1(a) and (b) illustrate the generic representation of the 2D and 3D convolutions, respectively. To perform a convolution, the convolution kernel slides over the domains of the input feature maps and generates the output feature maps. The 2D convolution extracts the independent features from a sequence of images with several 2D convolution kernels sliding along the M and N axis as shown in Fig. 1(a). In contrast, the 3D convolution captures both spatial and temporal information using a 3D convolution kernel sliding along the M, N and S axis as shown in Fig. 1(b). In 3D convolution, for given S input feature maps with the size of $M \times N$, the pixel cubes are extracted from the input feature maps and then convolve with the 3D convolution kernel. The pixel value at the position (m, n, s), on the output feature map, is given by [19]

$$o^{mns} = \sum_{p=0}^{P-1} \sum_{q=0}^{Q-1} \sum_{r=0}^{R-1} w^{pqr} i^{(m+p)(n+q)(s+r)} \quad (1)$$

where P, Q and R represent the height, width and length of the kernel, respectively. Besides, w^{pqr} shows the value of the kernel at the position (p, q, r), and for p, q and r equal to zero, i^{mns} is the pixel value of s^{th} input feature map at the position (m,n).

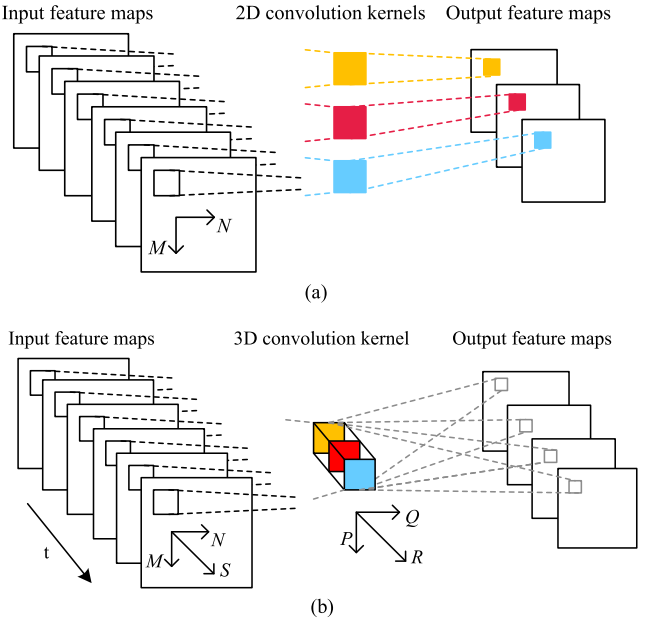


FIGURE 1. Generic representation for (a) 2D and (b) 3D convolutions. The small square in the input feature maps indicates the area extracted from the input feature maps and weighted by the kernel. To perform a convolution, this area is sliding pixel by pixel. Each slide generates one pixel in the output feature map indicated by the small squares in the output feature maps. It is necessary to slide the 3D convolution kernel along the time axis, S, to achieve the output feature map for each slide.

```
Loop_height    : for (row=0; row<M-P+1; row++){
  Loop_width   : for(col=0;col<N-Q+1;col++){
    Loop_length : for (to=0; to<S-R+1; to++){
      Loop_kernel_length : for( r =0; r < R; r ++){
      Loop_kernel_height : for( p =0; p < P; p ++){
        Loop_kernel_width : for( q=0; q < Q; q++){
          output [to][row][col]+=
      w[r][p][q]×input[to + r][row + p][ col + q]
      }}}}}}
```

FIGURE 2. Pseudo code of the *valid* 3D convolution.

Three convolution styles, *valid*, *same* and *full*, operate based on different boundary processing methods [20]. Here, we use a *valid* 3D convolution operation, whose convolution kernel is only allowed to visit the domains where the kernel is contained entirely within the input feature maps. For this 3D convolution style, the number of the output feature maps is (S-R+1), with the size of ($M-P+1$)×($N-Q+1$). Figure 2 presents the pseudo code of the *valid* 3D convolution.

Next, we evaluate the computational complexity of the 3D convolution represented by the number of multiplication and additional operations. To form a single output pixel using

TABLE 1. Computational complexity of the 3D convolution.

Elemental operations	Number of executions
Multiplication	$P\times Q\times R\times(S-R+1)\times(M-P+1)\times(N-Q+1)$
Addition	$(P\times Q\times R-1)\times(S-R+1)\times(M-P+1)\times(N-Q+1)$
Loading	$P\times Q\times R\times(S-R+1)\times(M-P+1)\times(N-Q+1)$

a $P \times Q \times R$ kernel, we have $P \times Q \times R$ multiplications, $(P \times Q \times R - 1)$ addition operations and $P \times Q \times R$ input pixels loading. For *valid* convolution, we may calculate the total number of the operations. Table 1 presents the computational complexity of the three elemental operations in a 3D convolution. As a result, to convolve simple feature maps with a small kernel, over one million operations are required. Therefore, to achieve real-time performance, i.e. thirty feature maps per second, a computational power of several GOPS is required. In fact, the presence of an extra, temporal, dimension in 3D convolution introduces massive operations.

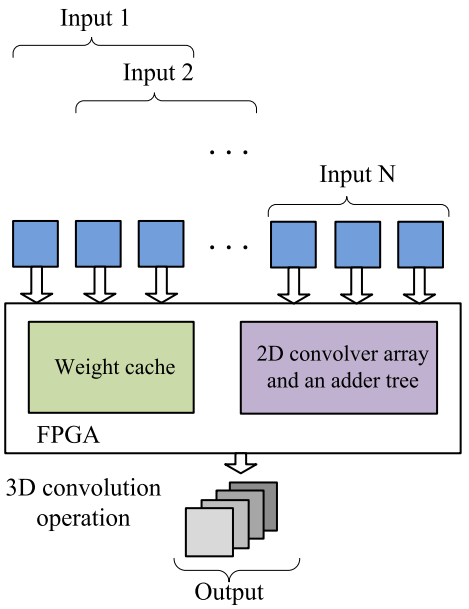


FIGURE 3. Operation principle of the basic 3D convolution with $R = 3$. Some feature maps are required to be loaded several times.

III. OPTIMIZATION AND IMPLEMENTATION

The 3D convolution operation in Fig. 1(b) may be represented as the summation of $R\times$ 2D convolution operations. In an FPGA accelerator, the 3D convolution operation may be achieved by $R\times$ 2D convolvers, same as 2D convolution operators, plus an adder tree. Figure 3 shows the operation principle of the basic 3D convolution. Due to the limited resources of the FPGA-based 2D convolvers, the pixels will not be permanently stored in the FPGA storage resources. As a consequence, in a 3D convolution operation formed

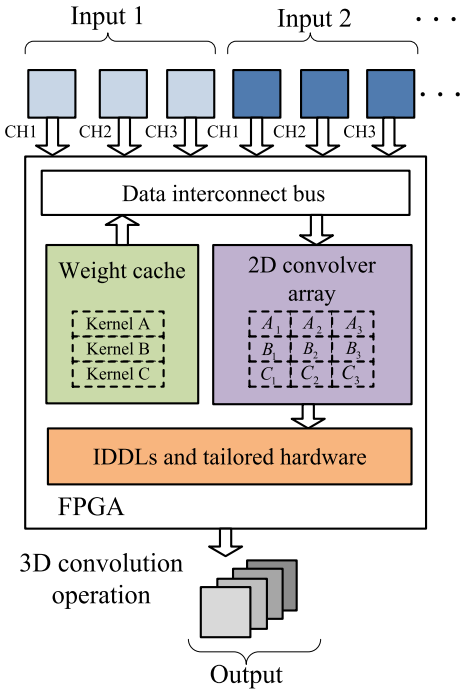


FIGURE 4. Schematic of the proposed 3D convolution with $R = 3$. The IDDLs in FPGA are used to avoid reloading of the input feature maps.

by $R\times$ 2D convolvers, the input feature maps are required to be repeatedly loaded. The loading repetition takes much time which highly affects the acceleration performance, and consequently, enforces reducing the loading times of the data transmission.

A. OPTIMIZATION OF 3D CONVOLUTION OPERATION

Figure 4 depicts the general schematic of the 3D convolution using Intermediate Data Delay Lines (IDDL). The IDDLs in an FPGA are utilized to temporally store the intermediate data, to avoid the feature map loading repetition. As a particular case, a kernel with the length, R, of 3 is shown in Fig. 4, where w^{pq1}, w^{pq2} and w^{pq3} are denoted as A, B and C. Three input channels, CH1, CH2 and CH3 are interconnected with three 2D kernels A, B and C. Besides, nine 2D convolvers are required, each of which labelled by X_i, where $X \in \{A, B, C\}$, denoting the kernel convolved with the input feature maps. The subscript $i = 1, 2, 3$ represents the channel in which the input feature map is loaded. Once the three input feature maps are loaded, each 2D convolver provides an output, and then, nine 2D convolution outputs are generated. We define x_i^j as the 2D convolution output of the j^{th} feature map convolving with X_i, where $j = i + 3n$, $S - 3 < 3n < S$, $\forall n \in N$ and $x \in \{a, b, c\}$.

Loading the first three feature maps, by adding a_1^1, b_2^2 and c_3^3, provides the first 3D operation result. IDDL_1 stores the sum of a_2^2 and b_3^3, while IDDL_2 stores a_3^3. Besides, since b_1^1, c_1^1 and c_2^2 are not used, the three 2D convolvers B_1, C_1 and C_2 are disabled by the tailored hardware, which will be explained in the next subsection. Then, loading the fourth, fifth and sixth input feature maps, adding c_1^4 and $a_2^2 + b_3^3$

TABLE 2. Comparison between the loading times of the basic and optimized 3D convolution operation.

Operation	Load Times
Basic operation	$(S-2R+3)\times R\times M\times N$
Optimized operation	$S\times M\times N$

which has been stored in the $IDDL_1$, provides the second 3D convolution operation result. Then, the third result may be achieved by adding b_1^4, c_2^5 and the delay-line a_3^3. The summation of a_1^4, b_2^5 and c_3^6 represents the fourth 3D convolution operation result. $IDDL_1$ stores the sum of a_2^5 and b_3^6, while $IDDL_2$ stores the a_3^6. Follow-up work is a similar repetition. The IDDLs provide the required data for the adders and also store a_3^j and the sum of a_2^j and b_3^j. The sum of a_2^j and b_3^j is stored in the $IDDL_1$, and then, adding with c_1^j provides an output feature map. Similarly, the delay-line a_3^j waits for the sum of b_1^j and c_2^j to make an addition.

The proposed 3D convolution operation, shown in Fig. 4, significantly reduces the data loading times. All input pixels are only required to be loaded once. Table 2 presents the comparison between the loading times in basic and optimized 3D convolution operations. If R reads three, the basic loading times are almost three times the optimization ones. The output feature map corresponds to the time loading, while the optimized operation may provide all corresponding output feature maps.

```
Loop_height : for (row=0; row<M-P+1; row++){
  Loop_width : for(col=0;col<N-Q+1;col++){
    Loop_length : for (to=0; to<S-R+1;
      to+=Kernel_Length){
#pragma HLS UNROLL
      For（too=to; too<min(to+Kernel_Length,S-
        R+1);too++）{
        Loop_kernel_length : for( r =0; r < R; r ++){
          Loop_kernel_height : for( p =0; p < P; p ++){
            Loop_kernel_width : for( q=0; q < Q; q++){
              output [to][row][col]+=
              W[r][p][q]×input[to + r][row + p][ col + q] }}}}
      }}}
```

FIGURE 5. Pseudo code of the proposed optimized 3D convolution operation.

Figure 5 presents the pseudo code of the proposed optimized 3D convolution operation. It may be seen that the Loop-length is divided into small loops, where the number of the loops is determined by the length of the 3D convolution kernel.

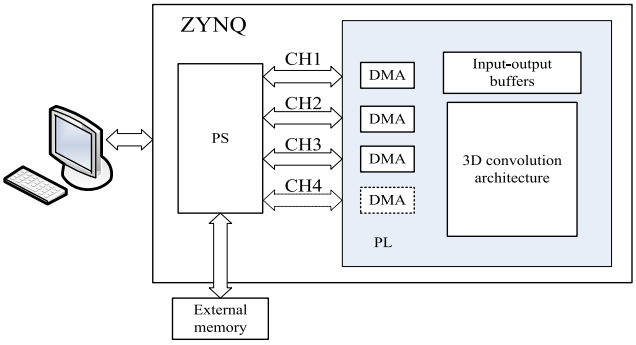


FIGURE 6. The implementation architecture for the optimized 3D convolution accelerator, presenting 3 or 4 channel feature maps at one time depending on the length of the 3D convolution kernel.

B. OPTIMIZED 3D CONVOLUTION ACCELERATOR IN FPGA

Figure 6 illustrates the data flowchart of the optimized 3D convolution accelerator based on the FPGA. We implemented the proposed accelerator in a ZYNQ chip, which is a programmable SOC from Xilinx. The number of the transmission channels in the Processing System (PS) and Programming Logic (PL) is equal to the length of the convolution kernel, which is considered to be either 3 or 4. Prior to the data processing, all input feature maps are stored in an external memory by PS. To calculate the 3D convolution, three or four, input feature maps are loaded to the PL for one time, and the pixels of the input feature maps are loaded in the raster scan. The 3D convolution operation is performed in the PL, which is consists of three parts, including Direct Memory Access (DMA), input-output buffer and 3D convolution architecture. As soon as the DMA module receives the data streams of the input feature maps, input buffers temporarily store the data streams. Then, the input buffers transmit the data streams to the 3D convolution architecture. Next, the architecture provides the output feature maps by convolving the input data streams. Then, the output feature maps are transmitted to the PS through the DMA modules, while they are temporarily stored in the output buffers.

Figure 7 presents the optimized 3D convolution architecture. The proposed 3D convolution is composed of various segments including the kernel caches, an array of 2D convolvers, IDDLs, adders, a data output controller and data interconnect bus. The kernel caches permanently store the 3D convolution kernels, to be accessible for the 3D convolution architecture. Each 2D convolver is directly connected to an input channel.

Figure 8 shows the details of the 2D convolvers, used in the 3D convolution in Fig. 7, where a full buffer scheme is adopted [13]. In this scheme, $(P\text{-}1)$ FIFOs with the length of $(N\text{-}Q)$ are employed to temporally hold the data before they reach to the 2D convolvers. We use P sets of right shifters, each of which consists of Q registers to assemble the $P\times Q$ convolution windows. The window operation is the sum of inner pixels multiplied by the corresponding weights of the kernel. Once a new pixel is loaded, the convolution window

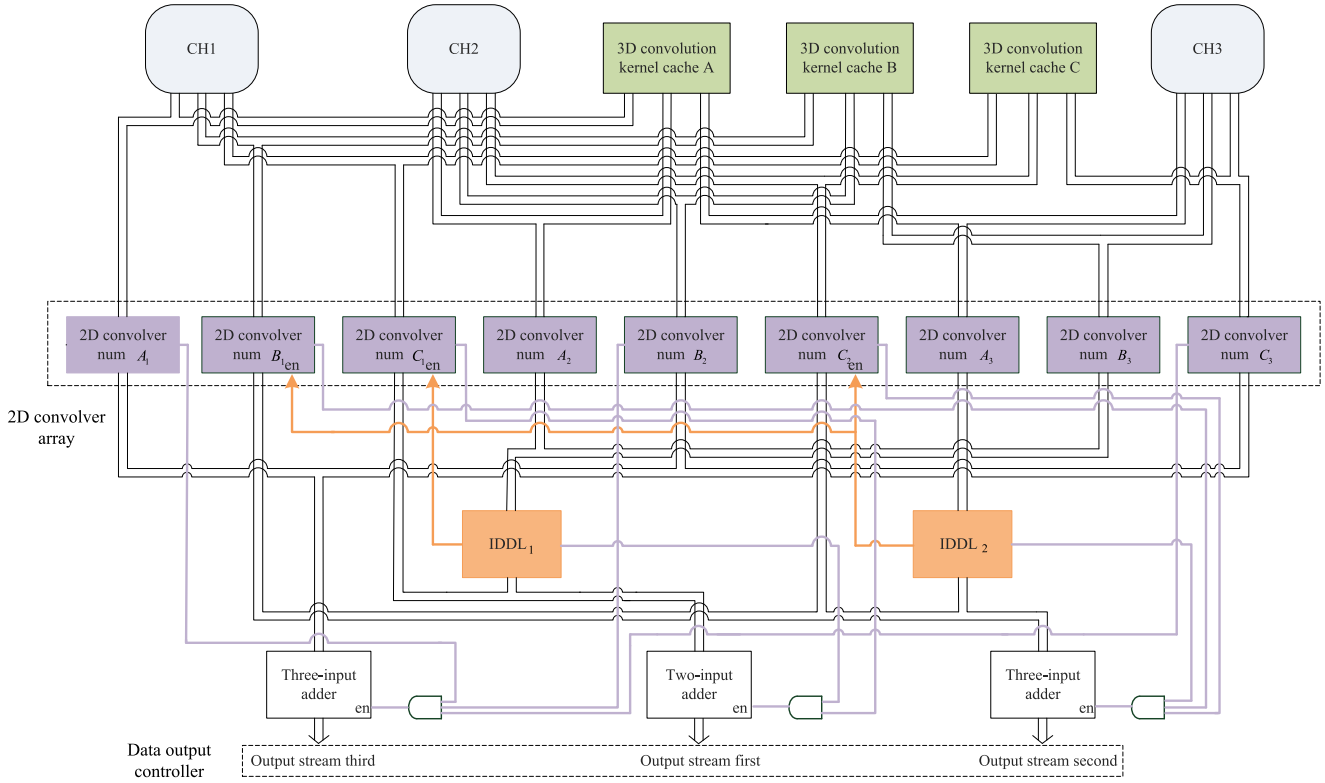


FIGURE 7. 3D convolution architecture composed of the kernel caches, an array of 2D convolvers, IDDLs, data interconnect bus, adders and a data output controller.

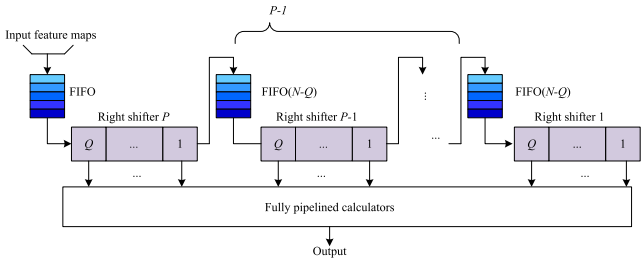


FIGURE 8. 2D convolver architecture, composed of the FIFOs, right shifters and fully pipelined calculators.

automatically moves to the next position. To accomplish the parallel operations, the loop kernel height and weight are fully unrolled and pipelined, which may significantly enhance the throughput.

Moreover, in Fig. 7, the IDDLs are used to avoid loading repetition of the input feature maps, where the length of the IDDLs reads $(M-P+1)\times(N-Q+1)$. The IDDLs are shift registers and may be implemented with flip-flops or block-RAMs depending on the specific FPGA device. The current outputs of the 2D convolvers are stored in the IDDLs until the loading of the next feature map. To reduce the power consumption, the 2D convolvers are kept off except those whose corresponding IDDLs are full. The data interconnection bus is the data transfer path between the various parts of the structure. The adders sum the relevant outputs and are switched off unless they read valid inputs. This design provides $(S-R+1)$ output feature maps, while simultaneously, reduces the power consumption. The adders have different priorities determined by the data output controllers. In this architecture, the adder which is connected to IDDL_1 has the highest priority. The arrangement of the priority levels provides a basis for PS, judging the order of the output feature maps.

IV. EXPERIMENTAL DEMONSTRATION

This section experimentally demonstrates the performance of the implemented accelerator. First, we show the resource usage of the accelerator under different pixel precisions. Then, we present the performance of the designed structure, and compare it with other proposed technologies.

Figure 9 shows the photograph of the realized accelerator using the Xilinx ZC706 developing board. This board is composed of the Xilinx ZYNQ, itself consists of a Kintex-7 FPGA and a dual ARM Cortex-A9 Processor, and 1GB DDR3 memory with the frequency bandwidth up to 4.2GB/s. The results are achieved using the Xilinx Vivado-2016.1 developing software. The design specifications are as follows. The input signal is a 30f/s low resolution video, where the resolution of each frame is 256×256. Various pixel precisions are chosen, i.e. 8, 10, 12, 16, 32-bit signed and 32-bit float. The kernel lengths are chosen as 3 and 4, where the kernel height×weight are considered as 3×3, 5×5, 7×7, 9×9 and 11×11. Table 3 lists the resource

TABLE 3. FPGA resource usage for the designed 3D convolution processor prototype.

Parameter		Length = 3				Length = 4			
Precision	Kernel Size	BRAM_18K	DSP48E	FF	LUT	BRAM_18K	DSP48E	FF	LUT
8-bit signed	3×3	73	27	2238	2877	108	36	2995	3856
	5×5	79	75	5118	5034	116	100	4856	6732
	7×7	85	147	9219	7929	124	196	12307	10592
	9×9	91	243	14484	12726	132	324	19342	16978
	11×11	97	363	21039	17328	140	484	28146	23124
10-bit signed	3×3	89	27	2490	3033	132	36	3340	4084
	5×5	95	75	5775	5328	140	100	7722	7124
	7×7	101	147	10674	8324	148	196	14252	11144
	9×9	107	243	16923	13449	156	324	22574	17954
	11×11	113	363	24732	18384	164	484	32996	25534
12-bit signed	3×3	105	27	2775	3225	156	36	3710	4314
	5×5	111	75	6765	5622	164	100	9042	7496
	7×7	117	147	12329	8817	172	196	16458	11778
	9×9	123	243	20241	14313	180	324	26988	19096
	11×11	129	363	29571	19368	188	484	39440	25834
16-bit signed	3×3	137	54	5475	4923	228	72	7320	6574
	5×5	143	150	13947	10332	236	200	18602	24392
	7×7	149	294	27231	18291	244	392	36318	40684
	9×9	155	486	43827	30492	252	648	58440	40658
	11×11	161	726	64674	43275	260	968	86234	57720
32-bit signed	3×3	265	108	7497	6180	396	144	10003	8240
	5×5	271	300	19536	12789	404	400	26059	17057
	7×7	277	588	37947	22116	412	784	50604	29497
32-bit float	3×3 float	265	216	17838	39111	396	288	23792	52148
	5×5 float	271	600	46371	95358	404	800	61832	127164

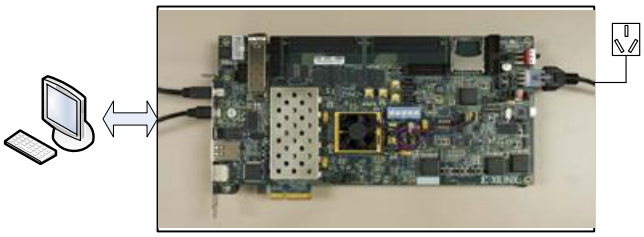


FIGURE 9. Photograph of the implemented prototype using Xilinx ZC706.

utilization of different pixel precisions with various kernel sizes. Flip-flops and look-up-tables represent the most basic resources in an FPGA. Moreover, the BRAM-18K and DSP48E are the storage and computation resources, respectively. The bigger the kernels, the higher the pixel precision, the more resources are utilized. For the pixel precision of less than 16 bits, one pixel multiplication is performed with one DSP48E. Considering the 16-bit signed pixel precision, a pixel multiplication is performed with two DSP48Es, while for the pixel precision of 32-bit signed and 32-bit float, a pixel multiplication is performed with four and eight DSP48Es. Due to the resource constraints, here, we only present the situations, which are implementable with the 32-bit signed and 32-bit float pixel precisions.

Let us now compare the proposed FPGA-based 3D accelerator with the CPU and GPU. The CPU platform is Intel Dual Core i7-6700K CPU, at 4GHz with a 32GB RAM. The GPU platform is characterized as NVIDIA GTX1080, possessing 2560 CUDA cores with an 8GB GDDR5 256-bit memory. The operating system for CPU and GPU is Ubuntu 16.04 with Keras deep learning software framework library. With the Keras framework, we may easily select the CPU or GPU to run the 3D convolution. The precision of the pixels and kernels in software is 32-bit float. A 100MHz system clock is considered for the FPGA-based 3D accelerator.

The latencies of various pixel precision have several clocks difference, which may be hided with the pipelined design

TABLE 4. Performance comparison of the FPGA, CPU and GPU.

Parameter		FPGA				CPU				GPU			
Kernel Size	Total Operations (MOP)	Comput. Time(ms)	Power Cons.(W)	GOPS	GOPS/w	Comput. Time(ms)	Power Cons.(W)	GOPS	GOPS/w	Comput. Time(ms)	Power Cons.(W)	GOPS	GOPS/w
3×3×3	97.55	6.84	0.175	14.26	81.49	44.32	32	2.2	0.069	9.39	44	10.38	0.236
5×5×3	266.72	6.98	0.198	38.21	192.98	62.28	32	4.28	0.134	11.46	44	23.27	0.529
7×7×3	514.5	7.11	0.235	72.36	307.91	83.12	32	6.19	0.193	11.63	44	44.24	1.005
9×9×3	836.95	7.25	0.265	115.44	435.62	102.85	32	8.14	0.254	11.89	44	70.39	1.6
11×11×3	1230.17	7.37	0.33	166.92	505.82	125.76	32	9.78	0.254	15.65	44	78.61	1.787
3×3×4	125.42	5.47	0.183	22.93	125.3	48.62	32	2.58	0.081	9.25	44	13.56	0.308
5×5×4	342.92	5.58	0.218	61.46	281.93	73.81	32	4.65	0.145	10.17	44	33.72	0.766
7×7×4	661.5	5.69	0.245	116.26	474.53	97.08	32	6.81	0.212	10.98	44	60.25	1.369
9×9×4	1076.07	5.8	0.288	185.53	644.2	121.85	32	8.83	0.276	14.27	44	75.41	1.714
11×11×4	1581.65	5.9	0.353	268.07	759.41	148.63	32	10.64	0.332	15.65	44	101.06	2.297

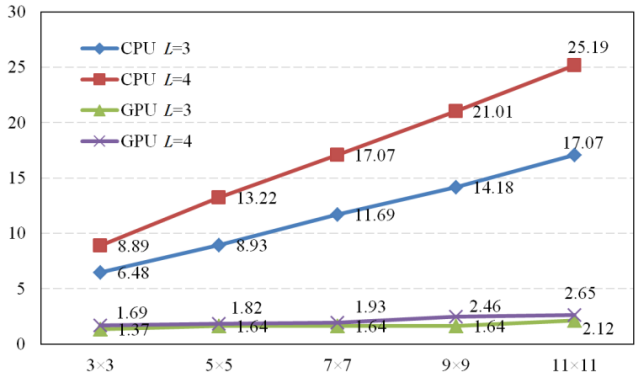


FIGURE 10. The GOPS ratio of the proposed accelerator over the CPU and GPU, with the length, *L*, of 3 and 4.

instead of totting-up when calculating the 3D convolution. Therefore, the computation time difference of the same kernel size with different precision can be less than one percent. Since the calculation of the same kernel size under different precisions consumes a same computation time, Table 4 only lists the experimental results with various kernel sizes. As we see in this table, The FPGA-based 3D accelerator deals with the thirty gray feature maps only in 5.9 ms, with the size of 256×256 convolving with a 11×11×4 kernel and reaches the speed of 268.07GOPS. The FPGA-based 3D accelerator presents a computational performance 14 times faster than that of the CPU and slightly faster than that of the GPU in average. Figure 10 provides the details of the acceleration specifications. The GOPS ratios of the proposed accelerator over the CPU and GPU increase with the kernel size. The maximum GOPS of the proposed convolution accelerator is 25.19× and 2.65× that of the CPU and GPU with the 11×11×4 kernel. The performance of the GOPS per watt for FPGA significantly outperforms that of the CPU and GPU.

Our work achieves the speed of one pixel per clock and 182f/s. For the computational complexity of 3D convolution is *R*, namely the length of the kernel, times that of 2D convolution, the calculation performance of our work is maximum four times that of the 2D convolution accelerators in work [13]–[16] and 3.75 times that of the work [17].

V. CONCLUSION

We presented an efficient 3D convolution operator based on the FPGA accelerator. The proposed structure significantly improves the convolution performance. The accelerator is characterized by an array of parallel 2D convolvers interconnected with IDDLs or adders, and other special-purpose hardware. The key attribute of the proposed processor is the implementation of the IDDLs to avoid the loading repetition of the processing feature maps. The FPGA-based accelerator presents the speed of 268.07GOPS and slightly faster than that of the fastest commercially available GPU. In future, we plan to apply the proposed FPGA-Based 3D accelerator to accelerate the 3D CNN models. The proposed accelerator may provide a higher throughput in a larger model.

REFERENCES

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[2] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[3] 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.

[4] 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-Programm. Gate Arrays*, 2015, pp. 161–170.

[5] W. Jiang, Y. Chen, H. Jin, R. Zheng, and Y. Chi, "A novel GPU-based efficient approach for convolutional neural networks with small filters," *J. Signal Process. Syst.*, vol. 86, no. 2, pp. 313–325, Mar. 2016.

[6] S. Potluri, A. Fasih, L. K. Vutukuru, F. Al Machot, and K. Kyamakya, "CNN based high performance computing for real time image processing on GPU," in *Proc. Joint 3rd Int. Workshop Nonlinear Dyn. Synchronization (INDS), 16th Int. Symp. Theor. Elect. Eng. (ISTET)*, Jul. 2012, pp. 1–7.

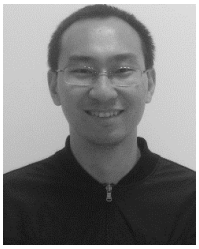
[7] T. Chen *et al.*, "DianNao: A Small-footprint High-throughput accelerator for ubiquitous machine-learning," in *Proc. 19th Int. Conf. Archit*, 2014, pp. 269–284.
[8] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2014, pp. 609–622.
[9] Z. Du *et al.*, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit*, 2015, pp. 92–104.
[10] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Proc. IEEE 31st Int. Conf. Comput. Design*, Oct. 2013, pp. 13–19.
[11] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," in *Proc. 21st Asia South Pacific Design Autom. Conf.*, Jan. 2016, pp. 575–580.
[12] K. Ovtcharov, O. Ruwase, J. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," Microsoft Research, Washington, DC, USA, White Paper 2(11), 2015.
[13] H. Zhang, M. Xia, and G. Hu, "A multiwindow partial buffering scheme for FPGA-based 2-D Convolvers," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 54, no. 2, pp. 200–204, Feb. 2007.
[14] S. Di Carlo, G. Gambardella, M. Indaco, D. Rolfo, G. Tiotto, and P. Prinetto, "An area-efficient 2-D convolution implementation on FPGA for space applications," in *Proc. 6th Int. Design Test Workshop*, Dec. 2011, pp. 88–92.
[15] G. Deepak, R. Mahesh, and A. Sluzek, "Design of an area-efficient multiplierless processing element for fast two dimensional image convolution," in *Proc. 13th IEEE Int. Conf. Electron. Circuits Syst.*, Dec. 2006, pp. 467–470.
[16] Z. B. Ma, Y. Yang, Y.-X. Liu, and A. A. Bharath, "Recurrently decomposable 2-D convolvers for FPGA-based digital image processing," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 63, no. 10, pp. 979–983, Oct. 2016.
[17] L. Rao, B. Zhang, and J. Z. Zhao, "Hardware implementation of reconfigurable 1D convolution," *J. Signal Process. Syst.*, vol. 82, no. 1, pp. 1–16, Jan. 2016.
[18] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Proc. Int. Conf. Artif. Neural Netw.*, 2014, pp. 281–290.
[19] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.
[20] I. Goodfellow, Y. Bengio, and A. Courville, "Convolutional networks," in *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, pp. 347–358.

HAI WANG was born in 1976. He received the B.E. degree in communication engineering, the M.E. degree in communication and information system, and the Ph.D. degree in measurement and instrument from Xidian University, Xi'an, China, in 1998, 2003, and 2007, respectively.

He is currently an Associate Professor with Xidian University. His main research interests are the time and frequency measurements, the frequency and phase measurements methods based on FPGA and SOPC, electronic instrument design, and time-frequency signal processing.

MENGJUN SHAO received the B.S. degree in measurement and control technology and instrument from Xidian University, Xi'an, China, in 2014, where she is currently pursuing the M.S. degree. Her current research interests are pattern recognition, deep learning, and FPGA design for embedded vision system.

YAN LIU received the M.S. degree in mechanical engineering and the Ph.D. degree in instrument science and technology from Xi'an Jiaotong University, China, in 2009 and 2014, respectively. He is currently a Lecturer with the School of Electro-Mechanical Engineering, Xidian University, China. His current research interests include computer vision, pattern recognition, and image processing



WEI ZHAO received the Ph.D. degree in communication and information system from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2011. He joined the School of Electro-Mechanical Engineering, Xidian University, Xi'an, in 2011. From 2014 to 2015, he was a Guest Researcher with the National Institute of Standards and Technology, Boulder, CO, USA. His current research interests include VNA calibration and uncertainty analysis.