IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 19, 2020, accepted May 14, 2020, date of publication May 18, 2020, date of current version June 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2995330

A Power-Efficient Optimizing Framework FPGA Accelerator Based on Winograd for YOLO

CHUN BAO[1], TAO XIE[1], WENBIN FENG[2,3], LE CHANG[1], AND CHONGCHONG YU[1]

[1]School of Computer and Information Engineering, Beijing Technology and Business University, Beijing 100048, China
[2]State Key Laboratory of Coal Mine Safety Technology, Fushun 113122, China
[3]China Coal Technology and Engineering Group Shenyang Research Institute, Fushun 113122, China

Corresponding author: Tao Xie (timonxie@126.com)

This work was supported in part by the Beijing Natural Science Foundation under Grant 4202015, in part by the National Key Research and Development Program of China under Grant 2018YFC0807900 and Grant 2018YFC0807903, and in part by the Special Fund of the Science and Technology Innovation and Entrepreneurship of the China Coal Technology and Engineering Group under Grant 2018-2-MS017.

ABSTRACT Accelerating deep learning networks in edge computing based on power-efficient and highly parallel FPGA platforms is an important goal. Combined with deep learning theory, an accelerator design method based on the Winograd algorithm for the deep learning object detection model YOLO under the PYNQ architecture is proposed. A Zynq FPGA is used to build the hardware acceleration platform of a YOLO network. The Winograd algorithm is used to improve traditional convolution. In the FPGA, the numerous multiplication operations in the YOLO network are converted into addition operations, reducing the computational complexity of the model. The data of the original model are processed at a low fixed point, reducing the resource consumption of the FPGA. To optimize memory, a buffer pipeline method is proposed, which further improves the efficiency of the designed accelerator. Experiments show that compared with the acceleration of the YOLO model based on GPUs and other FPGA platforms, the proposed method not only optimizes FPGA resource usage but also reduces power consumption to 2.7 W. Additionally, the detection accuracy loss is less than 3%.

INDEX TERMS FPGA, deep learning, Winograd, YOLO, buffer pipeline.

I. INTRODUCTION

In recent years, convolutional neural networks (CNNs) have achieved great success in many fields of computer vision. With the increase in application demand and the complexity of application scenarios, the layers of CNN networks continue to deepen, and the computational complexity of deep learning models increases [1]. Deep learning plays an important role in face recognition, industrial part detection, autonomous driving and voice recognition [2]–[5]. Object detection and recognition are the most challenging tasks in deep learning. In these algorithms, the most representative networks include single-shot-multibox-detection (SSD) [6], Faster R-CNNs [7] and the you only look once (YOLO) [8] series; among these, the YOLO algorithm has faster and more accurate performance than the other methods.

In recent research, most object detection and recognition algorithms are carried out in graphics processing units (GPUs). Due to the large number of parallel computing units in CNN, the performance advantage is more prominent in CNN with a large number of repeated multiplication and addition operations. In edge computing, it is obvious that a server platform, which has a high power consumption, cannot meet the requirements of being small, operating quickly and consuming little power. Therefore, a large number of object detection and recognition platforms based on edge computing have been proposed. Among them, application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs) are notable [9], and FPGAs can solve the specificity problem of ASICs with the advantages of high parallelism, high flexibility and low power consumption [10]. Therefore, research on CNN acceleration with deep learning based on edge computing platforms centres on FPGAs. In terms of using high-level synthesis tools, Suda *et al.* [11] proposed the acceleration of fixed-point CNNs using the OpenCL framework and proposed a systematic method to minimize execution time under given resource constraints. However, the author adopted a kernel implementation scheme

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

similar to GPU separation, so he did not make use of the special OpenCL feature of FPGA kernel-to-kernel pipeline communication to achieve better computing throughput and minimum memory bandwidth. The OpenCL acceleration system designed by Ling *et al.* [12] greatly improved performance by caching all intermediate features on the chip and using the Winograd algorithm to reduce multiplication and accumulation of convolution. However, the data buffering scheme limited the maximum size of the input image to 224×224, which cannot be achieved with a low-power and low-cost SoC-FPGA. Regarding the optimization of FPGA accelerators, Ma *et al.* [13] reduced memory access and data exchange by optimizing the loops of a CNN, performing quantitative analysis and optimizing the convolution of the neural network design resources. On the VGGNet network, throughput reached 645.25 GOPS, and latency was reduced to 47.97 ms. However, they did not verify the acceleration performance in large networks. Zhang *et al.* [14] proposed a ping-pong structure for the buffer to achieve FPGA memory optimization to ensure fast data interaction, and they analysed the design space of the proposed FPGA accelerator with a roofline model to solve the problem of optimizing computing resources and bandwidth. In the research on accelerating CNNs with FPGAs, on-chip resources and bandwidth are the greatest challenges for FPGAs [15]. Making the best use of FPGA resources to accelerate CNNs is the most important problem to be solved This problem can be solved if the Winograd algorithm is introduced into the acceleration of FPGAs. Lu *et al.* [16], [17] added the Winograd algorithm to the operation of a CNN for the first time and exceeded the limit of an FPGA's computing units. They reduced the computational complexity of the CNN and proposed the concept of a line buffer to reuse data efficiently. In another work of Lu and Liang [18], they combined the Winograd algorithm with CNN sparsity to improve the performance of an FPGA accelerator. However, regarding Winograd algorithm acceleration [19], too little work has been devoted to specific accelerator design for object detection and recognition.

As a representative network for object detection and recognition, much research has been done on the FPGA acceleration of the YOLO model, and many achievements have been made. Nguyen *et al.* [20] used RTL circuit to accelerate the YOLOv2 algorithm, quantified network weight parameters to binary, and reduced the digital signal processor (DSP) consumption in FPGA acceleration. Although dynamic random-access memory (DRAM) access and power consumption were reduced through data reuse and dynamic random access, the power consumption of this accelerator was still 18.29 W. For edge computing, the power consumption needed to be further improved. In another YOLOv2 acceleration work, Nakahara *et al.* [21] combined binary networks and support vector machines (SVMs) in lightweight YOLOv2. They designed a complete process and achieved excellent performance. Although the detection speed of YOLO was accelerated by reducing the computational complexity, they did not consider the optimization of memory access.

Based on the previous YOLOv2 and Winograd algorithm acceleration research, this paper proposes a Winograd-based YOLO algorithm acceleration method, which reduces the computational complexity of the YOLO algorithm in FPGAs, and proposes an FPGA accelerator memory optimization algorithm, which reduces the computation time of FPGAs in accelerating the YOLO algorithm.

The main contributions of this work are summarized as follows:

(1) When accelerating the YOLO algorithm in an FPGA, we introduce the Winograd algorithm into YOLOv2. Due to the presence of large amounts of convolution operations in YOLOv2 and high-level synthesis (HLS) tools to implement convolution operations, the loops of multiplication operations are replaced by addition operations. The multiplier resources consumed by convolution calculations are greatly reduced. Under the condition that the model accuracy is 78.25%, the multiplier utilization rate of FPGA is greatly reduced.

(2) To improve the efficiency of data caching and processing, we propose a state-of-the-art buffer pipeline method. Pipeline optimization is carried out on the data cache, which is involved in every convolution operation of the accelerator. The timing analysis shows that latency can be reduced while completing the same tasks.

(3) In this paper, a new YOLOv2 accelerator based on the PYNQ architecture is proposed, and each convolution and pooling layer of YOLOv2 is accelerated on the low-power and highly parallel Zynq FPGA platform. Moreover, fixed-point processing is carried out for the data, and the 32-bit floating-point weight parameters are fixed to 16 bits. In addition, power consumption is reduced to 2.7 W. The problem of the high power consumption of deep learning object detection and recognition models based on edge computing is solved.

The rest of this paper is organized as follows: Section 2 reviews the background knowledge and theoretical support of CNNs and the Winograd algorithm. Section 3 introduces the structure, data flow and fixed points of processing elements (PEs) based on the Winograd algorithm. Section 4 describes the pipeline structure of the accelerator PE's memory and timing analysis. Section 5 provides the overall architecture of the accelerator proposed in this paper. Section 6 presents our experiments and result analysis, and Section 7 concludes the paper.

II. BACKGROUND

A. WINOGRAD CONVOLUTION

As a type of artificial neural network, CNNs [22] can solve many problems that are difficult to solve with traditional neural networks. In terms of structure, a CNN usually includes a convolutional layer, activation function layer, pooling layer and fully connected layer, where the convolutional layer plays a key role [23]. However, due to the complicated calculations and large amount of data involved in the convolutional layer, the calculation time is great and many computational

resources are consumed. Therefore, the main focus of this paper is the convolutional layer calculation. In general, the typical input and output data of the convolutional layer include input feature maps, kernels and output feature maps. The results of the convolution operation between the input feature maps and convolution kernels are stored in output feature maps and then transferred to the next layer. Suppose there are N pieces of the input feature map and that its size is $H \times W$. The number of convolution kernels is M, and their size is $N \times K \times K$. In each convolution kernel, one of the dimensions must be equal to the number of pieces of the input feature maps [24]. In the convolution operation, the size of the kernels assigned to each feature map is $K \times K$. Suppose the stride is S; then, each $K \times K$ kernel operates once on each of the N feature maps and then sums the result as a unit element of the output feature map. Therefore, if the number of convolution kernels in the previous layer is M, then the number of feature maps in the next layer is M. If the size of the output feature map is $R \times C$, then $R = \frac{H-K}{S} + 1$ and $C = \frac{W-K}{S} + 1$. The operation of the convolutional layer is shown in equation (1).

$$Out[M][R][C] = \sum_{n=0}^{N} \sum_{i=0}^{K} \sum_{j=0}^{K} In[n][S \times r + i][S \times r + j] \times W[M][n][i][j] \quad (1)$$

where Out and In represent the 3D matrixes of the output and input feature maps and W represents the convolution kernel matrix [25].

The Winograd minimum filtering algorithm has a significant effect in reducing the amount of calculation needed for convolution with a smaller convolution kernel size [26]. For the YOLOv2 algorithm, the convolution kernels are all 3×3 and 1×1. These are so small that they are suitable for use with the Winograd algorithm to accelerate the convolution operation. By using $v(F(m, r)) = m + r - 1$ multiplications, the Winograd algorithm calculates the convolution kernel $F(m, r)$ with m outputs and dimensions [27]. Equation (2) indicates that the Winograd minimum filtering algorithm is used for the convolution operation when the convolution kernel size is 3 dimensions and the output matrix is 2 dimensions [16]. d_i represents the input feature map data, g_i represents the convolution kernel data, and m_i represents the output data.

$$\begin{aligned} F(2,3) &= \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_0 + m_1 + m_2 \\ m_1 - m_2 - m_3 \end{bmatrix} \\ m_0 &= (d_0 - d_2)g_0 \\ m_1 &= (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2} \\ m_2 &= (d_0 - d_1)\frac{g_0 - g_1 + g_2}{2} \\ m_3 &= (d_1 - d_3)g_2 \end{aligned} \quad (2)$$

The inputs of the Winograd minimal filtering algorithm are image data of $m - r + 1$ pixels, and the output is a vector of m dimensions. Because this algorithm performs 4 additions on the input data, 3 additions on the convolution kernel and 4 additions on the multiplied data, the algorithm increases the number of addition operations. However, the number of multiplications is reduced from 6 to 4 [28].

In the 2D convolution calculation, the convolution kernel size represented by $F(m \times m, r \times r)$ is $r \times r$, and the output matrix size is $m \times m$. $F(m \times m, r \times r)$ can be calculated from $F(m, r)$, as shown in equation (3).

$$F(m \times m, r \times r) = \begin{bmatrix} F(m,r)_{1,1} & \cdots & F(m,r)_{1,m} \\ F(m,r)_{2,1} & \cdots & F(m,r)_{2,m} \\ \vdots & \ddots & \vdots \\ F(m,r)_{m,1} & \cdots & F(m,r)_{m,m} \end{bmatrix} \begin{bmatrix} g_{r,1} \\ g_{r,2} \\ \vdots \\ g_{r,m} \end{bmatrix} \quad (3)$$

When the size of the output matrix is $m \times m$, the size of the input image data must be $(m + r - 1) \times (m + r - 1)$. The equations below can be used to calculate the number of multiplication and addition operations and the number of times the input data and the convolution kernel are processed.

Number of multiplications:

$$Multip = (m + r - 1) \times (m + r - 1) \quad (4)$$

Times the input data are processed:

$$Final_{ADD} = Mul \times Add_f + m \times Add_f \quad (5)$$

Times the convolution kernel is processed:

$$Filter_{ADD} = Mul \times Add_k + r \times Add_k \quad (6)$$

Number of additions:

$$Data_{ADD} = 2 \times Mul \times Add_d \quad (7)$$

Mul represents the number of multiplications required to compute $F(m, r)$. Add_f, Add_k and Add_d represent the total number of additions, the number of times the convolution kernel is processed and the number of times the input data used to calculate $F(m, r)$ are processed, respectively.

In conclusion, when 2D convolution is carried out using the Winograd algorithm, $F(m \times m, r \times r)$ can be obtained from $F(m, r)$, as shown in equation (8).

$$Out = A^T[(GFG^T) \odot (B^T InB)]A, \quad (8)$$

where the values of the transformation matrixes A, B and G can be determined by the values of m and r. Therefore, the transformation functions of the input and convolution kernel are:

$$Transform(In) = B^T InB, \quad (9)$$

$$Transform(F) = GFG^T. \quad (10)$$

The inverse transformation function is:

$$Inverse\_Transform(E) = A^T EA. \quad (11)$$

In the design of the Winograd PE in Section 3.1, the input convolution kernel parameters and feature map parameters can be converted rapidly before operation with the transformation matrix begins.

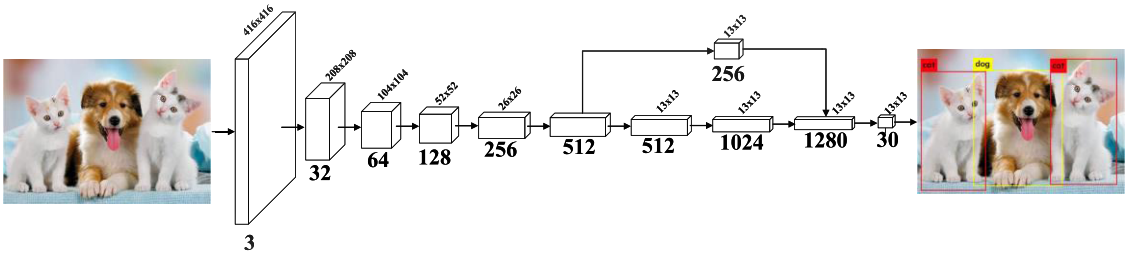


FIGURE 1. Network structure of YOLOv2.

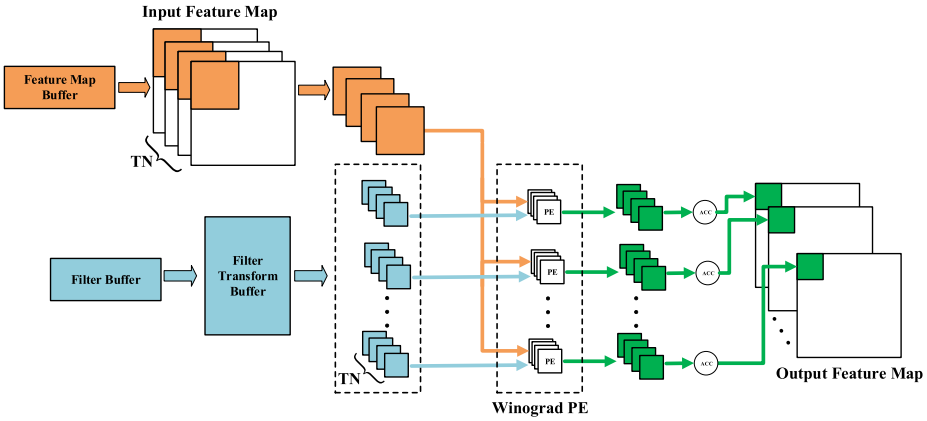


FIGURE 2. The data pipeline of the YOLOv2 accelerator.

B. YOLO NETWORK

The networks commonly used for object detection and recognition based on deep learning include R-CNN, the YOLO series, SSD, etc. Among these networks, SSD is widely used, but its structure is complex, the number of its parameters is large, and its speed is not as high as that of YOLO. YOLO has a similar structure to GoogleNet, but with a large increase in detection speed. On the server side, YOLOv2 performs well, but in the edge computing, the detection speed still has much room for improvement. To reflect the acceleration effect of FPGAs, we select the YOLOv2 network in the YOLO series, whose structure diagram is shown in Figure 1.

The framework used by YOLOv2 is the improved Darknet-19. The Darknet-19 network consists of nineteen convolutional layers, one average pooling layer, five maximal pooling layers and one softmax layer. In Darknet-19, there are a large number of 3×3 convolution filters, and in order to compress features and increase network depth, a 1×1 convolution filter was added between the 3×3 convolution filters. YOLOv2 is improved by removing the last convolutional layer, the average pooling layer and the softmax layer, adding three 3×3 convolutional layers and one 1×1 convolutional layer, and using convolution instead of a fully connected layer. A large number of convolution operations result in a longer runtime of the YOLOv2 model on embedded terminals. However, the FPGA's high-efficiency parallel computing capability can perform the series of convolution operations in YOLOv2.

III. PROPOSED YOLO ACCELERATOR BASED ON WINOGRAD

Combined with the background knowledge in Section 2, the Winograd algorithm was introduced into the convolution operations of the YOLO network when we designed the FPGA accelerator. In this section, we introduce the detail of the PE structure and fixed-point processing of low-point data for YOLO network acceleration.

A. WINOGRAD PE DESIGN

The Winograd algorithm achieves acceleration by reducing the number of multiplications, but the number of additions increases accordingly. Moreover, additional transformation calculations and storage for the transformation matrix are required. As the sizes of the convolution kernels and feature maps increase, the cost of addition, transformation and storage needs to be considered. Furthermore, the larger the feature map and transformation matrix are, the greater the loss of calculation accuracy. To solve this problem, we choose to only perform Winograd acceleration operations on the 3×3 convolution kernel and to group the input feature maps of the PE. To minimize the resource consumption and latency of storing the transformation matrix, we propose using the buffer pipeline optimization method.

The convolution operation of the YOLO model is shown in Figure 2. An input feature map entering the convolutional layer operation is stored in the on-chip buffer, and

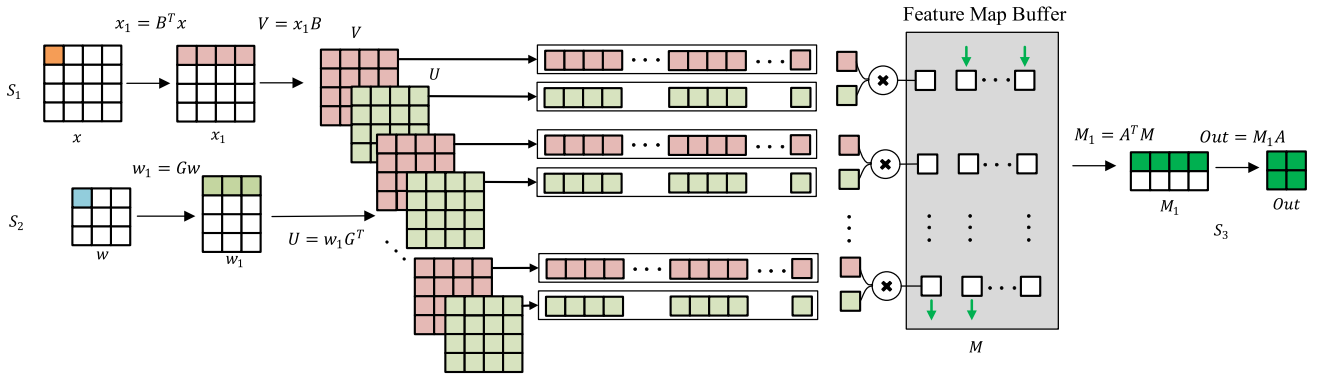


FIGURE 3. The Winograd PE structure of the YOLO v2 accelerator.

the model's parameter file is stored in the filter buffer. The feature map is expanded to obtain the feature map vector before N feature maps are transmitted to the Winograd PE operation unit. To maximize the parallelization of the operation, the model parameters and the feature map vectors are grouped separately. In the Winograd unit, the feature map vectors and the convolution kernels are multiplied and added. Finally, the convolution result of each feature map is obtained. The accumulator (ACC) unit fuses the features and stores the calculation result in the output feature map buffer. All operation results wait for the next process to be read.

The designed Winograd PE is divided into three parts that transfer the feature maps and kernel parameters to the convolution unit individually for calculation. As shown in Figure 3, this process can be divided into three steps: first we transform the feature map parameters taken from the buffer. As shown in Section 2.1, the values of the transformation matrixes A, B and G can be determined by the values of m and r. Thereby, the transformed transformation matrix V can be obtained. When the transformation of the feature map is complete, we take out the convolution kernel parameters that are stored in the buffer. In addition, the transformed feature matrix U can be obtained by using the transformation of Figure 3. For the last step, matrixes U and V are passed to the PE, and the value of matrix M can be obtained by the point multiplication operation. Finally, we obtain the output result. These three steps are labelled as S_1, S_2 and S_3 respectively.

Figure 4 shows the convolution operation pseudo-code of adding the Winograd algorithm into YOLO proposed in this paper. *Winograd* (in, F, out) represents the calculation function of Winograd designed for the FPGA. The size of the input feature map is $M \times H \times W$, and the size of the output feature map is $N \times R \times C$. When the data are input to the Winograd function for the acceleration operation, the feature map data and kernel data are expanded and grouped. In a conventional convolution operation, six cycles are executed. After the Winograd algorithm is added, Loop-5 and Loop-6 can be eliminated, which saves the multiplier computation caused by the Loop operation in the FPGA.

Pseudo-code of the Winograd Convolution Layer

```
in[M][H][W]:input images(M channels)
F[fi][fo],F[fi][fo][i][j]:weights
out[N][R][C]:output images(N channels)

for row = 0; row < H; row += 1 do          —Loop 1
  for col = 0; col < H; col++ do           —Loop 2
    for fi = 0; fi < M; fi++ do            —Loop 3
      for fo = 0; fo < N; fo++ do          —Loop 4
        for i = 0; i < K; i++ do           —Loop 5
          for j = 0; j < K; j++ do         —Loop 6
            out[row][col][fo] += F[fi][fo][i][j] * in[row+i][col+j][fi];
        } }
      Winograd(in[row][col][fi],F[fi][fo],out[row][col][fo])
} } } }

Winograd(in,F,out) {
    U = B^T inB
    V = GFG^T
    out = A^T[U□ V]A
}
```

FIGURE 4. Pseudo-code of Winograd. Winograd(in, F, out) is the Winograd function.

B. LOW-BIT FIXED POINT

In the calculation of a CNN, the sizes of the weights and bias parameters affect the performance of the whole network [29]. A large number of high-precision floating-point arithmetic operations will not only increase the power consumption of the model but also affect the computing speed. Therefore, before a CNN is used with an FPGA, the model data should be fixed-point. In a large number of studies [30]–[33], it was found that compared with floating-point number operations, fixed-point data are more suitable for efficient computation with FPGAs. In a study of Chen *et al.*, it was found that the power consumption of fixed-point data of 16 bits was only 0.136 times that of 32-bit data, while the accuracy of the model decreased by only 0.26%. Therefore, in training a YOLO model, 32-bit data was selected in this paper. However, when deploying the YOLO model in an FPGA, the feature map data, convolution kernels and bias parameters were quantified as 16 bits.

The fixed-point data process is shown in Figure 5. The 32-bit data consist of three parts [34], which are the sign bit (S, 1-bit), exponen bits (E, 8-bits) and mantissa bits

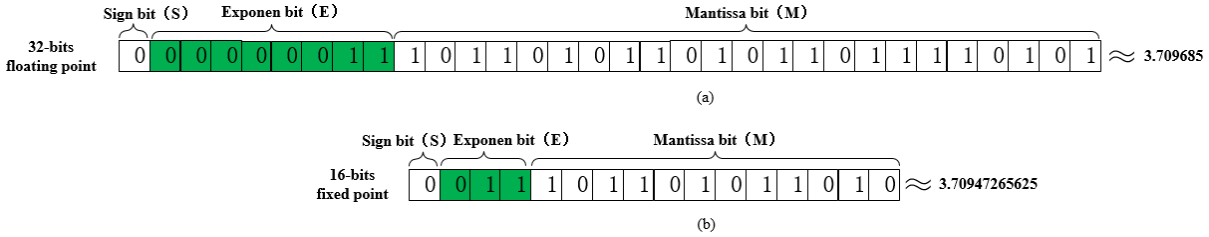


FIGURE 5. (a) 32-bit floating-point data (b) 16-bit fixed-point data.

(M, 23-bits). The exponen bits are the integer part of the floating-point number, and the mantissa bits are the fractional part of the floating-point number. Fixed-point numbers differ from floating-point numbers in that the decimal point is fixed. Once the three-part width of the fixed-point number is determined, the position of the decimal point will never change [35]. In Figure 5, the range of signed fixed-point numbers that can be expressed is $[-2^{E-1}, 2^{E-1}-1]$, and the data precision is 2^M.

We take the YOLO convolution filter data 3.709685 as an example. Its 32-bit floating-point number representation is shown in Figure 5(a); it contains 8 integer bits and 23 decimal bits, which is similar to the original data. Figure 5(b) shows the fixed-point data type used in this paper, which contains 1 sign bit, 3 integer bits, and 12 decimal bits; compared with the 32-bit floating-point data, the error is only 0.0002.

IV. MEMORY OPTIMIZATION BASED ON THE PROPOSED BUFFER PIPELINE

A. BUFFER PIPELINE

To solve the memory optimization problem of FPGA accelerator design, we propose the buffer pipeline method for the first time. In the logical part of Zynq, data interact with the CPU through an external storage DDR DRAM. The DDR is controlled by an advanced extensible interface (AXI) bus when data are exchanged with the accelerator. To ensure that the timing requirements are met in the data flow, a first-in first-out (FIFO) interface is added after the AXI bus. In this way, the data of the accelerator's input and output data can be transmitted efficiently.

At the input interface of the accelerator's PE, we usually add a buffer set to change the data format and wait for a certain length of time. The pipeline architecture of the accelerator proposed in this paper is shown in Figure 6. In the input data part of the accelerator, the input buffer sets are divided into three parts: Buf_In1, Buf_In2 and Buf_In3. The output buffer sets are divided as Buf_Out1, Buf_Out2 and Buf_Out3. This pipeline structure can take full advantage of each buffer to ensure data interaction and transmission. During the jump cycle of the clock bus, the storage capacity of each buffer can be maximized. The details of the timing advantages of this structure are analysed in Section 3.2.

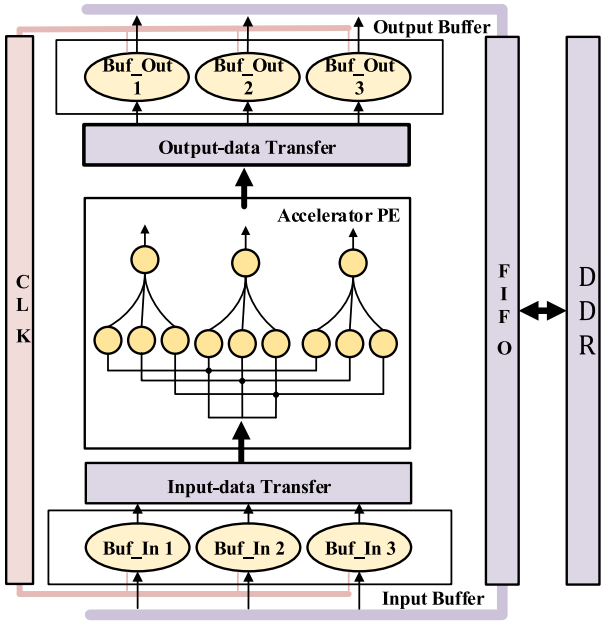


FIGURE 6. The internal structure of the accelerator based on the buffer pipeline optimization proposed in this paper. The clock lines of all the buffers are uniformly planned on the CLK bus, and the data in the buffer are formatted in the input data transfer module.

B. TIMING ANALYSIS OF THE BUFFER PIPELINE

Every Winograd PE operation inside the accelerator is cached through the buffer sets, including taking data from the buffer into the PE, computing the PE and taking the data for the cache from the PE. The time to input the data for each buffer is T_{in}, the time needed for the data from the buffer to enter the PE for computing is T_{co}. The time taken by the buffer after the operation of the acceleration PE is T_{out}. The time to complete the whole task flow is T_{task}. Assume that the number of tasks completed in the acceleration PE is n and that $T_{in} \neq T_{co} \neq T_{out}$ (it does not matter if all three operations take the same amount of time). According to the sequence of the regular memory access structure, the time to complete all tasks is found with equation (12).

$$T_{sum} = n \times T_{task} = n \times (T_{in} + T_{co} + T_{out}) \tag{12}$$

The proposed buffer pipeline structure improves the single buffer set to a three-buffer structure, and carries out a

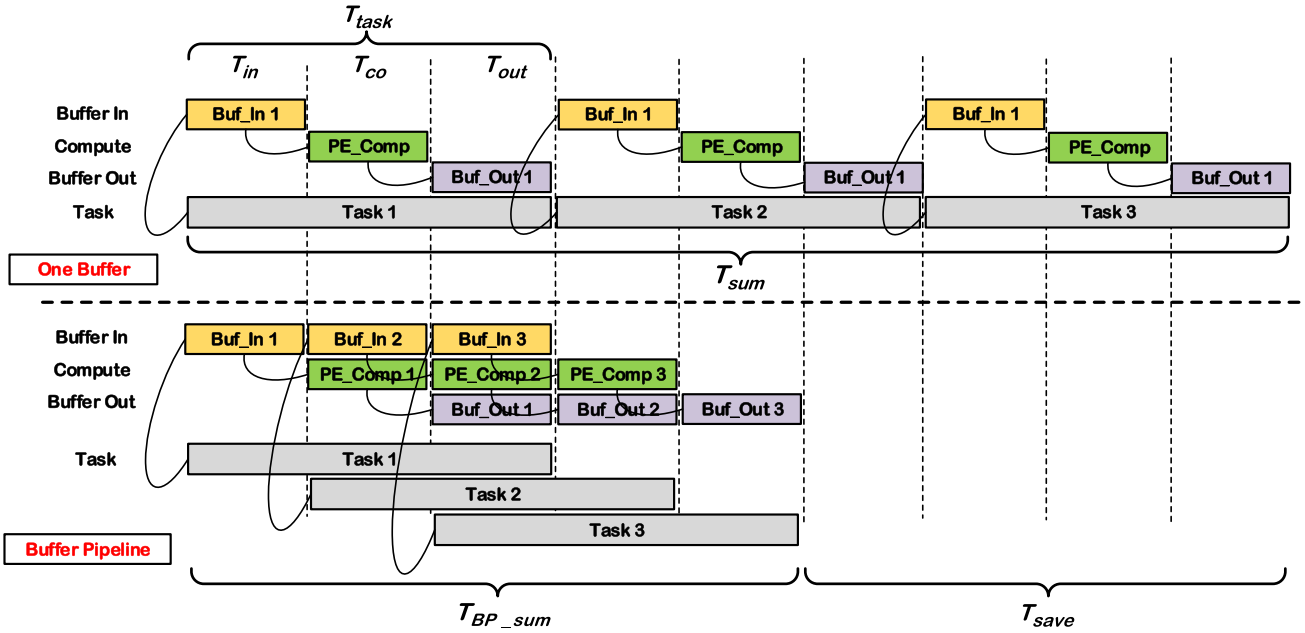


FIGURE 7. Timing changes when the buffer pipeline is not added to the accelerator computing unit and when the buffer pipeline is added. When the task is executed three times, the time the buffer pipeline method saves is T_{save}, where Buffer In, Compute, and Buffer Out represent the three stages of completing the computing task.

three-level pipeline. We set the following variables:

$$T_{\max}^{123} = \max(T_{in}, T_{co}, T_{out}), \tag{13}$$
$$T_{\max}^{12} = \max(T_{in}, T_{co}), \tag{14}$$
$$T_{\max}^{23} = \max(T_{co}, T_{out}). \tag{15}$$

Since the whole task can be divided into three stages, the total time latency when completing n tasks is shown in equation (16).

$$\begin{aligned} T_{BP\_sum} &= T_{in} + T_{\max}^{12} + T_{\max}^{123} \times [n-(3-1)] + T_{\max}^{23} + T_{out} \\ &= T_{in} + \max(T_{in}, T_{co}) + \max(T_{in}, T_{co}, T_{out}) \\ &\quad \times [n-(3-1)] + \max(T_{co}, T_{out}) + T_{out}. \end{aligned} \tag{16}$$

We take a set of three tasks as an example to make a timing diagram for the conventional calculation and the buffer pipeline structure as shown in Figure 7.

$$T_{sum} = 3 \times T_{task} = 3 \times (T_{in} + T_{co} + T_{out}) \tag{17}$$

When the buffer pipeline is used for memory, the time taken to complete the entire task is shown in equation (18).

$$\begin{aligned} T_{BP\_sum} &= T_{in} + T_{\max}^{12} + T_{\max}^{123} + T_{\max}^{23} + T_{out} \\ &= T_{in} + \max(T_{in}, T_{co}) + \max(T_{in}, T_{co}, T_{out}) \\ &\quad + \max(T_{co}, T_{out}) + T_{out}. \end{aligned} \tag{18}$$

According to the property of inequality, it can be found that:

$$\begin{aligned} &\max(T_{in}, T_{co}) + \max(T_{in}, T_{co}, T_{out}) + \max(T_{co}, T_{out}) \le 3T_{co} \\ &\max(T_{in}, T_{co}) + \max(T_{in}, T_{co}, T_{out}) + \max(T_{co}, T_{out}) < 3T_{in} \\ &\max(T_{in}, T_{co}) + \max(T_{in}, T_{co}, T_{out}) + \max(T_{co}, T_{out}) < 3T_{out} \end{aligned} \tag{19}$$

Therefore, clearly, $T_{sum} > T_{BP\_sum}$. The time saved by the method proposed in this paper is T_{save}, as shown in equation (20).

$$\begin{aligned} T_{save} &= T_{sum} - T_{BP\_sum} \\ &= n \times (T_{in} + T_{co} + T_{out}) - \{T_{in} + \max(T_{in}, T_{co}) \\ &\quad + \max(T_{in}, T_{co}, T_{out}) \times [n-(3-1)] \\ &\quad + \max(T_{co}, T_{out}) + T_{out}\} \end{aligned} \tag{20}$$

V. OVERALL ARCHITECTURE OF THE YOLO ACCELERATOR

The overall hardware architecture of the accelerator designed in this paper is shown in Figure 8. The software structure can be divided into two parts: PS (Processing System) and PL (Progarmmable Logic). On the PS side, it integrates ARM cores and uses the Linux operating system. In addition, the Python language environment is preserved when the operating system is ported. The CPU can control all interfaces between the PS and PL. The accelerator uses CPU scheduling to input the feature maps of the YOLO network parameters into the DDR buffer and interact with peripheral operating system circuits through the bus. The CPU uses the AXI bus to read the operation results of the acceleration circuit and execute the application of image pre-processing and display on the PS side. On the PL side, the data in the external DDR are cached in the on-chip RAM, and the convolution and pooling circuits of the YOLO accelerator are laid out and wired in the FPGA. The hardware design bitstream file and the design instruction file (Tcl) are passed to the OS overlay. The hardware circuit and the YOLO's IP core are parsed

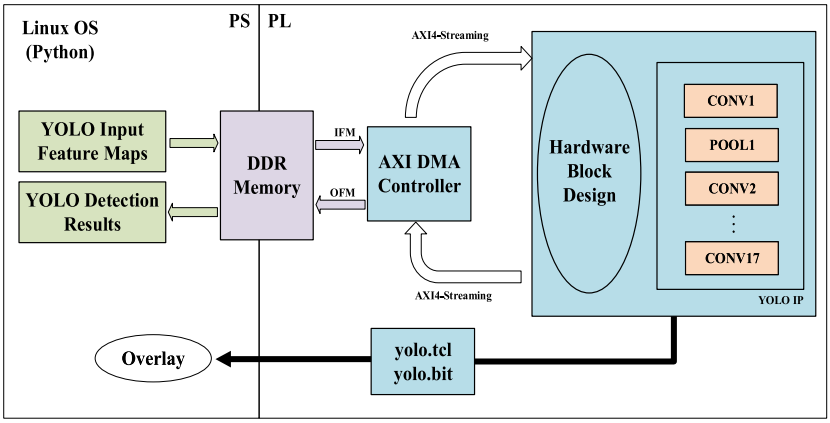


FIGURE 8. Overview of the YOLOv2 accelerator architecture based on the PYNQ platform.

in the overlay. Finally, the data path of the entire hardware accelerator is completed.

VI. EXPERIMENTAL EVALUATION

A. EXPERIMENTAL SETUP

The Winograd YOLO accelerator proposed in this work is generated by Vivado HLS 2018.2. The HLS can convert a C-language program into the RTL circuit required by the FPGA. The hardware architecture is built in Vivado 2018.2. The acceleration platform is developed in a Xilinx PYNQ-z2 board. The main chip is Zynq XC7Z020-1CLG400C, which contains a 630 KB block RAM, 220 DSP slices, an ARM dual-core Cortex-A9 processor, and an external 512 MB DDR3.

The configuration parameters of the YOLOv2 network selected for acceleration are shown in Table 1. Because the size of the YOLOv2 model convolution kernels are 3×3 and 1×1, it is suitable for the calculations of the Winograd algorithm.

B. FIXED-POINT DATA ANALYSIS

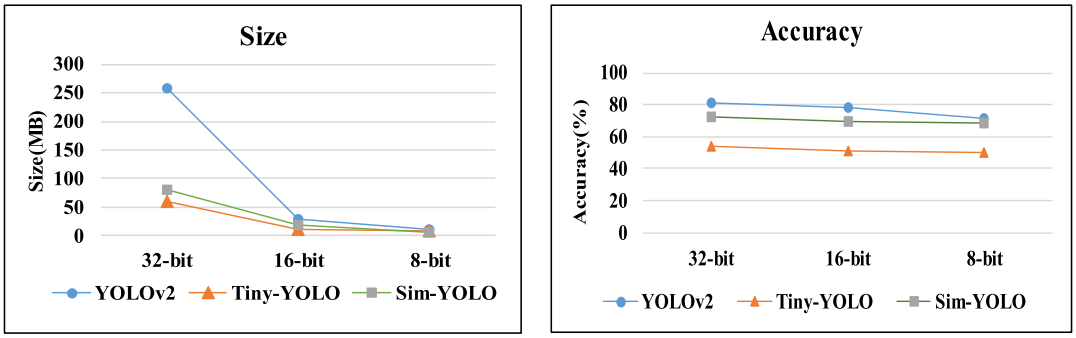
In Section 3.2, we proposed a method of quantifying 32-bit data as 16-bit data. In this section, we apply this method to YOLOv2, Tiny-YOLO [36], and Sim-YOLO. Tiny-YOLO and Sim-YOLO are simplified versions of YOLOv2. We train these three networks on the PASCAL VOC 2007+2012 dataset, and extract a convolution kernel and bias parameters. Then, the 32-bit data obtained undergo 16-bit and 8-bit fixed-point processing. In the fixed-point transformation of the 8-bit data, we set the sign bit (S, 1-bit), exponen bit (E, 2-bits) and mantissa bit (M, 5-bits). Then, we use the fixed-point network on the original test dataset for evaluation, and the final accuracy is shown in Figure 9(b). Compared with the original-precision network, the accuracy of the YOLOv2 network decreased by 2.88% at 16 bits. The accuracy of Tiny-YOLO and Sim-YOLO decreased by 2.91% and 2.1% respectively. At 8 bits, compared to the original-precision network, the accuracy of the YOLOv2 network drops by 8.32%. The accuracy of Tiny-YOLO and Sim-YOLO decreased by 4.12% and 3.25% respectively. In the YOLOv2 network, compared with using 8-bit fixed-point parameters, the accuracy of the network model decreases less with 16-bit fixed-point parameters. Therefore, we select 16 bits as the FPGA-accelerated network parameter type.

As we can see from Figure 9(a), in the process with fixed-point data, the storage occupied by the network parameters also decreases. Compared with the original-precision model, the size of the YOLOv2 model was reduced by 7 times in the 16-bit fixed-point transformation. Tiny-YOLO and Sim-YOLO are reduced in size by 8 and 12 times, respectively.

TABLE 1. Parameter configuration of the YOLOv2 model used in this paper.

Layer	Type	Filters	Size/Stride	Output
1	C	32	3×3/1	224×224×32
2	M		2×2/2	112×112×32
3	C	64	3×3/1	112×112×64
4	M		2×2/2	56×56×64
5	C	128	3×3/1	56×56×128
6	C	64	1×1/1	56×56×64
7	C	128	3×3/1	56×56×128
8	M		2×2/2	28×28×128
9	C	256	3×3/1	28×28×256
10	C	128	1×1/1	28×28×128
11	C	256	3×3/1	28×28×256
12	M		2×2/2	14×14×256
13	C	512	3×3/1	14×14×512
14	C	256	1×1/1	14×14×256
15	C	512	3×3/1	14×14×512
16	C	256	1×1/1	14×14×256
17	C	512	3×3/1	14×14×512
18	M		2×2/2	7×7×512
19	C	1024	3×3/1	7×7×1024
20	C	512	1×1/1	7×7×512
21	C	1024	3×3/1	7×7×1024
22	C	512	1×1/1	7×7×512
23	C	1024	3×3/1	7×7×1024
24	C	1000	1×1/1	7×7×1000

Note: C=Convolutional Layer, M=Maxpool Layer.



(a) (b)

FIGURE 9. (a) The size changes of the YOLOv2, Tiny-YOLO, and Sim-YOLO models under 32-bit, 16-bit, and 8-bit parameter types, respectively. (b) The accuracy changes of the YOLOv2, Tiny-YOLO, and Sim-YOLO models under 32-bit, 16-bit, and 8-bit parameter types, respectively.

TABLE 2. Comparison of the proposed design with previous works with YOLO hardware.

	Sim-YOLO-v2 on GPU [37]	Tiny-YOLO [37]	Lightweight YOLO-v2 [21]	Tiny-YOLO-v2 [20]	This work (YOLO-v2)
Platform	GTX Titan X	Zynq Ultrascale+	Zynq Ultrascale+	Virtex-7 VC707	Pynq-z2
Frequency	1 GHz	N/A	300 MHz	200 MHz	125 MHz
BRAMs	N/A	N/A	1706	1026	880
DSPs	N/A	N/A	377	168	153
LUTs-FFs	N/A	N/A	135 K-370 K	86 K-60 K	38 K-36 K
Image Size	416×416	416×416	224×224	416×416	416×416
Accuracy (mAP) (%)	66.79	48.5	67.6	51.38	78.25
Power (W)	170	6	N/A	8.7	2.7

Thus, the 16-bit fixed point parameters can not only ensure the accuracy of the YOLOv2 model but also reduce the size of the model.

C. RESULT ANALYSIS

In this experiment, the Winograd algorithm parameters are used in the convolutional layers of YOLOv2. The YOLO accelerator is generated by Vivado HLS. In the block design, the hardware bit file and parameter file are generated. The operating system PS schedules the hardware logic and allocates the acceleration resources. Before the network parameters are loaded into the FPGA, the data are quantized to the fixed-point 16-bit type. The final average time for the accelerated platform to process each image is 124 ms, and the average detection accuracy is 78.25%. The experimental environment and detection results are shown in Figure 10.

At the end of the experiment, we compare the acceleration effect of this accelerator with other platforms. As shown in Table 2, compared with the GPU platform, this accelerator based on the PYNQ platform not only maintains accuracy but also greatly reduces power consumption. In comparison with the accelerator implemented on the Zynq Ultrascale+ platform, after the introduction of the Winograd algorithm, the number of adders in our accelerator increases, but the number of DSPs is significantly reduced. In addition, overall resource consumption is reduced. In this experiment,

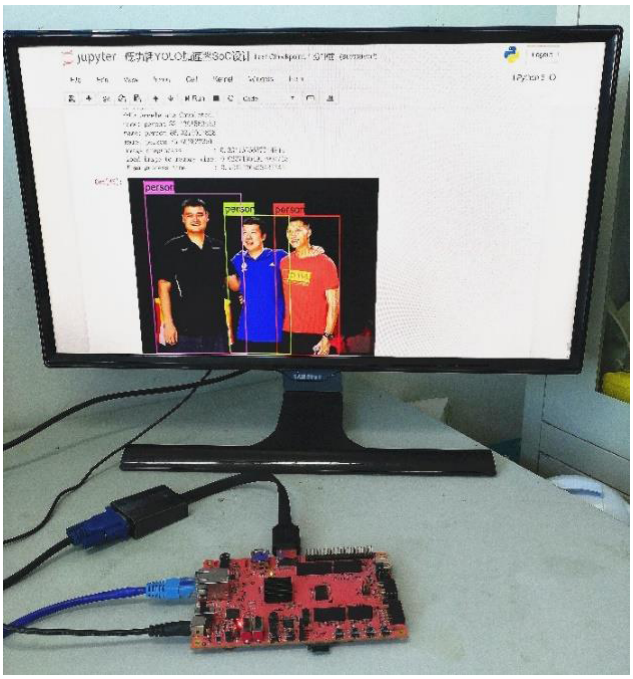


FIGURE 10. Experimental environment and detection results.

accuracy is improved because the YOLOv2 model that we select in this work has greater precision than simplified YOLO networks such as Tiny-YOLOv2.

VII. CONCLUSION

To solve the problem of limited resources and excessive power consumption in edge computing for deep learning detection networks, we proposed an architecture based on the Winograd algorithm for YOLO accelerator design on an FPGA platform, which quantified network parameters and solved the problem that a YOLO network needs a large number of DSPs for FPGA acceleration. To optimize memory, we proposed a buffer pipeline optimization method, which greatly improved the efficiency of data interaction. Test results show that compared with the implementation of YOLO in a GPU and in Zynq Ultrascale+, the method proposed in this work maintained accuracy, saved resources, greatly reduced power consumption and had a profound effect on accelerating deep learning networks in edge computing.

REFERENCES

[1] D. Wang, K. Xu, Q. Jia, and S. Ghiasi, "ABM-SpConv: A novel approach to FPGA-based acceleration of convolutional neural network inference," in *Proc. 56th Annu. Design Autom. Conf.*, Las Vegas, NV, USA, Jun. 2019, pp. 1–6.

[2] R. He, X. Wu, Z. Sun, and T. Tan, "Wasserstein CNN: Learning invariant features for NIR-VIS face recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 7, pp. 1761–1773, Jul. 2019.

[3] H. Zhang, T. Xu, M. Elhoseiny, X. Huang, S. Zhang, A. Elgammal, and D. Metaxas, "SPDA-CNN: Unifying semantic part detection and abstraction for fine-grained recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 1143–1152.

[4] O. Abdel-Hamid, A. R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," in *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014, doi: 10.1109/TASLP.2014.2339736.

[5] X. Z. Chen, K. Kundu, Y. K. Zhu, A. Berneshawi, H. M. Ma, S. Fidler, and R. Urtasum, "3D object proposals for accurate object class detection," in *Proc 28th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Montreal, QC, Canada, 2015, pp. 424–432.

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," 2015, *arXiv:1512.02325*. [Online]. Available: http://arxiv.org/abs/1512.02325

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," 2015, *arXiv:1506.01497*. [Online]. Available: http://arxiv.org/abs/1506.01497

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015, *arXiv:1506.02640*. [Online]. Available: http://arxiv.org/abs/1506.02640

[9] Z. Zhu, J. Zhang, J. Zhao, J. Cao, D. Zhao, G. Jia, and Q. Meng, "A hardware and software task-scheduling framework based on CPU+FPGA heterogeneous architecture in edge computing," *IEEE Access*, vol. 7, pp. 148975–148988, Sep. 2019.

[10] M. Zhao, C. Hu, F. Wei, K. Wang, C. Wang, and Y. Jiang, "Real-time underwater image recognition with FPGA embedded system for convolutional neural network," *Sensors*, vol. 19, no. 2, p. 350, Jan. 2019.

[11] 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)*, Monterey, CA, USA, 2016, pp. 16–25.

[12] A. C. Ling, U. Aydonat, S. O'Connell, D. Capalija, and G. R. Chiu, "Creating high performance applications with Intel's FPGA OpenCL SDK," in *Proc. 5th Int. Workshop OpenCL (IWOCL)*, Toronto, ON, Canada, 2017, p. 1.

[13] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, Monterey, CA, USA, 2017, pp. 45–54.

[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-Program. Gate Arrays (FPGA)*, Monterey, CA, USA, 2015, pp. 161–170.

[15] X. Hu, Y. Zeng, Z. Li, X. Zheng, S. Cai, and X. Xiong, "A resources-efficient configurable accelerator for deep convolutional neural networks," *IEEE Access*, vol. 7, pp. 72113–72124, 2019.

[16] L. Q. Lu, Y. Liang, Q. C. Xiao, and S. G. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *Proc. IEEE Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Napa, CA, USA, Apr. 2017, pp. 1–8.

[17] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 857–870, Apr. 2020.

[18] L. Lu and Y. Liang, "SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2018, pp. 1–6.

[19] Y. Huang, J. Shen, Z. Wang, M. Wen, and C. Zhang, "A high-efficiency FPGA-based accelerator for convolutional neural networks using winograd algorithm," *J. Phys., Conf. Ser.*, vol. 1026, no. 1, May 2018, Art. no. 012019.

[20] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.

[21] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Monterey, CA, USA, Feb. 2018, pp. 31–40.

[22] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[23] I. Rocco, R. Arandjelovic, and J. Sivic, "Convolutional neural network architecture for geometric matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 11, pp. 2553–2567, Nov. 2019.

[24] Y. Li and Y. Du, "A novel software-defined convolutional neural networks accelerator," *IEEE Access*, vol. 7, pp. 177922–177931, 2019.

[25] X. Wu, R. He, Z. Sun, and T. Tan, "A light CNN for deep face representation with noisy labels," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 11, pp. 2884–2896, Nov. 2018.

[26] J. Yu, Y. Hu, X. Ning, J. Qiu, K. Guo, Y. Wang, and H. Yang, "Instruction driven cross-layer CNN accelerator with winograd transformation on FPGA," in *Proc. Int. Conf. Field Program. Technol. (ICFPT)*, Melbourne, VIC, Australia, Dec. 2017, pp. 227–230.

[27] X. Wang, C. Wang, and X. Zhou, "Work-in-progress: WinoNN: Optimising FPGA-based neural network accelerators using fast winograd algorithm," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Turin, Italy, Sep. 2018, pp. 1–2.

[28] X. Liu, J. Pool, S. Han, and W. J. Dally, "Efficient sparse-winograd convolutional neural networks," 2018, *arXiv:1802.06367*. [Online]. Available: http://arxiv.org/abs/1802.06367

[29] F. U. D. Farrukh, T. Xie, C. Zhang, and Z. Wang, "Optimization for efficient hardware implementation of CNN on FPGA," in *Proc. IEEE Int. Conf. Integr. Circuits, Technol. Appl. (ICTA)*, Beijing, China, Nov. 2018, pp. 88–89.

[30] D. Larkin, A. Kinane, and N. O'Connor, "Towards hardware acceleration of neuroevolution for multimedia processing applications on mobile devices," in *Neural Information Processing (ICONIP)* (Lecture Notes in Computer Science), vol. 4234. Berlin, Germany: Springer, Oct. 2006, pp. 1178–1188.

[31] C. Farabet, Y. LeCun, K. Kavukcuoglu, E. Culurciello, B. Martini, P. Akselrod, and S. Talay, "Large-scale FPGA-based convolutional networks," *Mach. Learn. Very Large Data Sets*, vol. 9780521192248, pp. 399–419, May 2011.

[32] 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. ASPLOS*, Feb. 2014, vol. 49, no. 4, pp. 269–284.

[33] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Cambridge, U.K., Dec. 2014, pp. 609–622.

[34] Q. Chen, C. Xin, C. Zou, X. Wang, and B. Wang, "A low bit-width parameter representation method for hardware-oriented convolution neural networks," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, Guiyang, China, Oct. 2017, pp. 148–151.

[35] L. Z. Lai, N. Suda, and V. Chandra, "Deep convolutional neural network inference with floating-point weights and fixed-point activations," Mar. 2017, *arXiv:1703.03073*. [Online]. Available: https://arxiv.org/abs/1703.03073
[36] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," Dec. 2016, *arXiv:1612.08242*. [Online]. Available: https://arxiv.org/abs/1612.08242
[37] T. B. Preuser, G. Gambardella, N. Fraser, and M. Blott, "Inference of quantized neural networks on heterogeneous all-programmable devices," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2018, pp. 833–838.

CHUN BAO received the B.S. degree in automation from Liaoning Shihua University, Fushun, China, in 2017. He is currently pursuing the M.S. degree in control theory and control engineering with Beijing Technology and Business University.

His research interests include deep learning, FPGA acceleration, and edge computing.

TAO XIE received the Ph.D. degree in traffic information engineering and control from Beijing Jiaotong University, Beijing, China.

He is currently a Lecturer with Beijing Technology and Business University. His research interests include weak signal detection and processing, FPGA acceleration, and electronic circuit design in extreme environments.

WENBIN FENG received the B.S. degree from the China University of Mining and Technology.

He is currently a Research Fellow with the China Coal Technology and Engineering Group Shenyang Research Institute. His research interests include artificial intelligence, spectral analysis, and analytical instruments.

LE CHANG received the B.S. degree in communications engineering from Beijing Technology and Business University, Beijing, China, in 2018, where he is currently pursuing the M.S. degree in control engineering.

His research interests include image processing and edge computing.

CHONGCHONG YU received the Ph.D. degree in computer science from the University of Science and Technology Beijing, Beijing, China.

She is currently a Professor with Beijing Technology and Business University. Her research interests include artificial intelligence, machine learning, and pattern recognition.