A Block-Based and Highly Parallel CNN Accelerator for Seed Sorting

Xiaoting Sang, Zhenghui Hu, Huanyu Li, Chunlei Li, and Zhoufeng Liu

1School of Electronic and Information Engineering, Zhongyuan University of Technology, Zhengzhou 450007, China
2Hangzhou Innovation Institute, Beihang University, Hangzhou 310051, China
3College of Oceanography and Space Informatics, China University of Petroleum East China, Qingdao 266580, China

Correspondence should be addressed to Zhenghui Hu; zhenghuihu2013@163.com

Received 27 August 2022; Revised 27 October 2022; Accepted 31 October 2022; Published 17 November 2022

Seed sorting is critical for the breeding industry to improve the agricultural yield. The seed sorting methods based on convolutional neural networks (CNNs) have achieved excellent recognition accuracy on large-scale pretrained network models. However, CNN inference is a computationally intensive process that often requires hardware acceleration to operate in real time. For embedded devices, the high-power consumption of graphics processing units (GPUs) is generally prohibitive, and the field programmable gate array (FPGA) becomes a solution to perform high-speed inference by providing a customized accelerator for a particular user. To date, the recognition speeds of the FPGA-based universal accelerators for high-throughput seed sorting tasks are slow, which cannot guarantee real-time seed sorting. Therefore, a block-based and highly parallel MobileNetV2 accelerator is proposed in this paper. First, a hardware-friendly quantization method that uses only fixed-point operation is designed to reduce resource consumption. Then, the block convolution strategy is proposed to avoid latency and energy consumption increase caused by large-scale intermediate result off-chip data transfers. Finally, two scalable computing engines are explicitly designed for depth-wise convolution (DWC) and point-wise convolution (PWC) to develop the high parallelism of block convolution computation. Moreover, an efficient memory system with a double buffering mechanism and new data reordering mode is designed to address the imbalance between memory access and parallel computing. Our proposed FPGA-based MobileNetV2 accelerator for real-time seed sorting is implemented and evaluated on the platform of Xilinx XC7020. Experimental results demonstrate that our implementation can achieve about 29.4 frames per second (FPS) and 10.86 Giga operations per second (GOPS), and 0.92× to 5.70× DSP-efficiency compared with previous FPGA-based accelerators.

1. Introduction

Seed sorting plays an essential role in seed production, processing, and marketing, which not only determines the survival rate of seeding and the yield rate of production but also has an important impact on subsequent product processing. However, many researchers only realize the recognition accuracy required in applications through simulation experiments [1]. Therefore, the recognition speed is insufficient for high-throughput seed-sorting industrial equipment. Early manual sorting methods are inefficient, subjective, and have high error rates. With the development of automatic agricultural production, machine vision technologies have been widely used in agricultural product sorting and quality grading [2]. The traditional machine learning method characterizes the handcrafted features such as the color, shape, and texture of seeds by image processing operations. Then, classifiers such as support vector machines (SVMs) [3], linear discriminant analysis (LDA) [4], and artificial neural networks (ANNs) [5] are selected for seed classification. However, the diversity of defective seeds makes the model difficult to distinguish fine-grained differences, resulting in low classification accuracy.

Recently, CNNs have been applied to various applications such as image classification, object detection, and natural language understanding [6, 7]. With their powerful feature learning ability, CNNs have also made some breakthroughs in agricultural fields, such as seed sorting. To
achieve this, Kozlowski et al. [8] adopt a CNN model to recognize the barley varieties and achieve 93% classification accuracy. Huang et al. [9] apply VGGNet19 [10] and GoogLeNet [11] to recognize the defective corn kernels. Khan et al. [12] adopt fine-tuned VGG-S and AlexNet to extract deep features and design a multihead SVM classifier to detect fruit diseases, which reaches an accuracy of over 97%. However, the network models, as mentioned above, are relatively complex in extracting high-level semantic features, which hinders the deployment of the CNN model on embedded terminal devices with limited resources. Thus, searching for suitable CNNs for embedded devices is crucial.

The lightweight CNN is a recently emerging trend to realize deployment on embedded devices. Deep separable convolution and grouped convolution (GC) are two commonly used operations in lightweight networks, which have the advantages of reducing parameters and computation. Representative lightweight networks, such as SqueezeNet [13], MobileNet [14, 15], and ShuffleNet [16], can not only apply on a PC but also on an embedded terminal, occupying small memory of the embedded devices. They provide a strong technical support for seed sorting based on embedded device development and deployment.

GPU, a mainstream acceleration platform for deep learning training, has an incomparable advantage in processing speed. However, GPU’s natural high-power consumption inevitably limits its application in edge computing scenarios, where low power and low latency are required strictly. In contrast, FPGA, as a parallel computation-intensive acceleration hardware platform, has the characteristics of low power dissipation and high efficiency, which is a suitable platform for automatic seed sorting. In addition, FPGA architectures are more flexible and can handle irregular parallelism for emerging deep CNNs and custom-defined data types. Despite its advantages, current FPGA-based hardware acceleration still suffers many problems due to limited memory bandwidth and on-chip resources. On the one hand, many parameters involving inputs/outputs and intermediate computation results lead to inevitable off-chip storage. FPGA devices struggle to provide enough memory bandwidth to achieve adequate computational parallelism [17, 18]. On the other hand, most accelerators adopt tiling in the spatial and channel dimensions to accommodate large-scale feature maps with limited on-chip storage. This operation introduces an incredible amount of off-chip data transfers and data processing in the host processor during inference, leading to an increase in latency and energy consumption [18, 19]. In addition, the researchers mainly focus on the design of compute-intensive CNN. General-purpose CNN accelerators are not efficient for lightweight models, which is reflected in the difficulty of achieving the theoretical peak performance of hardware.

To attack these problems, a block-based and highly-parallel CNN accelerator for seed sorting is designed and implemented by using high-level synthesis (HLS). The advantage of our design over existing accelerators is the proposed hardware-friendly block convolution strategy, which reduces power consumption and computational latency. This paper designs an innovative data reordering mode to provide efficient data access for computational engines, which is neglected by most accelerators. More prominently, the accelerator proposed is highly scalable, adapting to FPGA of different capacities and CNNs with various parameter configurations. The main contributions of this paper are summarized as follows:

(i) A hardware-friendly quantization method is designed to reduce the data bit width of the model, which minimizes the memory consumption of seed sorting parameters on FPGA.

(ii) The block convolution strategy is proposed to relieve the pressure of on-chip memory caused by significant parameters in the MobileNetV2 model, which could avoid latency and energy consumption increase caused by large-scale intermediate result off-chip data transfers.

(iii) An efficient memory system with a double buffering mechanism and new data reordering mode is proposed to design a fantastic acceleration architecture. In addition, two scalable computing engines are customized for DWC and PWC.

(iv) The proposed architecture and optimizing strategies are implemented with the MobileNetV2 accelerator on the resource-constraint XC7020 FPGA platform. Different types of seeds are recognized, and high FPS and DSP-efficiency are achieved, which provide practical solutions and a theoretical support for the mechanization and automation of seed sorting.

The remained structure of this paper is organized as follows. Section 2 briefly introduces the related work. Section 3 describes the proposed method. Section 4 elaborates on the design of an FPGA-based MobileNetV2 accelerator, focusing on the computing module’s storage policy and acceleration method. Section 5 provides comprehensive experimental verification and discussion, and finally, the conclusions of this paper are presented in Section 6.

2. Related Work

2.1. Development of Seed Sorting with CNNs. Numerous types of research have demonstrated that using CNN as a generic extractor can significantly improve the accuracy of crop classification tasks compared with traditional feature engineering methods [20, 21]. Xu et al. [22] developed a wheat recognition system based on VGGNet16 with a classification accuracy of 98.19%, which could adequately distinguish 40 different wheat grain varieties. Zhu et al. [23] designed their own CNN based on the ResNet model for the characteristics of cotton seed varieties to extract features of seven cotton seed varieties. Various algorithms such as partial least squares discriminant analysis (PLS-DA), logistic regression (LR), and support vector machine models are used in their study to classify the seeds, achieving an accuracy of 80%. Altuntaş et al. [24] applied VGGNet16 to automatically identify haploid and diploid maize seeds through a transfer learning approach. In reference [25], a CNN-ANN model is used to classify corn seeds, and 2250 instances are tested in
26.8 seconds with a classification accuracy of 98.1%. Dong et al. [26] used the pruning VGG16 network model to improve the inference speed by 2.1 and 2.8 times on red kidney bean and corn seed datasets, which achieve 97.38% and 96.56% seed sorting accuracy, respectively. Shi et al. [27] compared numerous algorithms for an electronic nose in identifying liquors, in which the chaotic BPNN is utilized to perform pattern recognition tasks, ultimately achieving a convergence speed 75.5 times faster than that of the BPNN.

CNN-based seed sorting methods have gained excellent performance improvement. However, the large-scale CNNs used in previous works not only occupy a large amount of memory space but also suffer from high computational complexity, which is unsuitable for real-time processing. To issue this, some works [28] adopt CNNs with small parameters and low computational complexity, but their accuracies are not satisfactory. Additionally, even though some other works [29, 30] design their models with a few parameters, they still require too much memory space due to the dense connection, which is unsuitable for real-time industrial processing.

### 2.2. Lightweight Method for Seed Sorting

The computational complexity, high-power consumption, and high memory utilization hinder the implementation of computation-intensive CNN on embedded systems. The number of parameters and operations needed for the typical computation-intensive CNN is shown in Table 1. Take VGGNet16 [10], for example, it has more than 138 million parameters, 527 MB of storage space, and 34.09 GOPS operations, of which the convolution layer accounts for 10% and the FC layer accounts for the last 90%. Embedded devices cannot process such large amounts of data, but network lightweight makes this possible.

The lightweight CNN is a recently emerging trend to realize deployment on embedded devices. Numerous studies have shown that lightweight networks represented by SqueezeNet [13], MobileNet [14, 15], and ShufuNet [16] can extract detailed information and the ability to discriminate background of seed sorting is complex, making it difficult to achieve relatively low resource utilization. Most of the accelerators focus only on the compute-intensive CNNs while the flexibility is poor. Motamed et al. [44] combine three parallel computing optimization methods: inter-output parallelism, inter-convolution kernel parallelism, and intra-convolution kernel parallelism, which only apply to FPGA with abundant resources. Furthermore, some FPGA-based CNN accelerators focus only on optimizing the computational resources and memory access of FPGA-based CNN accelerators.

However, hardware acceleration based on FPGA still presents many challenges due to memory bandwidth and on-chip resources. For example, Li et al. [43] improve the throughput rate of the accelerator significantly by designing an end-to-end CNN accelerated method, but the flexibility is poor. Motamed et al. [44] not only focus on optimizing the computational resources but also propose a space exploration framework that can evaluate and explore various architectural options by optimizing the computational resources and memory access of FPGA-based CNN accelerators.

### 2.3. FPGA-Based CNN Accelerators

In recent years, various FPGA-based CNN accelerators have been proposed. As demonstrated in reference [36], bandwidth is the main bottleneck that affects performance when the data reuse rate is low, and FPGA hardware resources are not fully utilized. Many CNN accelerators are based on data reuse to increase bandwidth upper limit. An accelerator adopting data reuse and task parallelization is designed in [37] to improve per DSP throughput under constant memory bandwidth. Literature [38] designed a data allocation scheme to maximize the burst length of each transaction in the external memory and the fully connected layer and avoid unnecessary access delay. References [39, 40] introduce the Winograd algorithm to optimize computational efficiency by using a circular unfolding and tiling strategy, which makes the designed CNN accelerator performance improve. Some other works [41, 42] not only focus on optimizing the computational resources but also propose a space exploration framework that can evaluate and explore various architectural options by optimizing the computational resources and memory access of FPGA-based CNN accelerators.

### 3. The Proposed Seed Sorting Method

#### 3.1. Lightweight CNN Architecture for Seed Sorting

The background of seed sorting is complex, making it difficult to extract detailed information and the ability to discriminate between different seeds is low. Traditional CNNs have

<table>
<thead>
<tr>
<th>Network</th>
<th>Parameters (million)</th>
<th>Operation (GOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGGNet16</td>
<td>138.35</td>
<td>34.09</td>
</tr>
<tr>
<td>GoogleLeNet</td>
<td>6.97</td>
<td>3.16</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>25.5</td>
<td>7.72</td>
</tr>
<tr>
<td>FCN8s</td>
<td>18.64</td>
<td>43.03</td>
</tr>
</tbody>
</table>

Table 1: Characteristics of compute-intensive CNNs.
disadvantages, such as single-scale feature extraction and scattered regions of interest. Therefore, this paper selects the lightweight network MobileNetV2 as the feature extraction network according to the seeds’ characteristics. The specific structure is shown in Figure 1 and Table 2. Bottleneck residual block is the core part of the MobileNetV2 network, which is composed of DWC and PWC. DWC is to filter each feature map in the input channel and extract the shape, contour, and other complex features of the seed image. PWC preserves more detailed texture information, such as seed edges, corners, and colors, by increasing the dimension of the input feature channels. Moreover, the nonlinear activation function Relu6 immediately following the PWC layer is improved to linear operation, which preserves the diversity of seed feature information and enhances the expression ability of the target features. The bottleneck residual block adopts the independent double-branch structure. The shallow features are introduced into the subsequent layers in series splicing, which integrates the features with different receptive fields and improves the reuse of shallow features in the network. Stacking the same topological structure can enlarge the receptive field of the feature map and extract advanced semantic features, thus increasing the accuracy.

This model uses depth-wise separable convolution instead of standard convolution to reduce model computation [15], which is more conducive to deployment to edge devices with limited resources. MobileNetV2 adds an extra batch normalization (BN) between the convolutional layer and the

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input</th>
<th>Operator</th>
<th>Expansion factor</th>
<th>Output channel</th>
<th>Repeat time</th>
<th>Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2242 × 3</td>
<td>conv2d</td>
<td>—</td>
<td>32</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1122 × 32</td>
<td>Bottleneck</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3–4</td>
<td>1122 × 16</td>
<td>Bottleneck</td>
<td>6</td>
<td>24</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5–7</td>
<td>562 × 24</td>
<td>Bottleneck</td>
<td>6</td>
<td>32</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>8–11</td>
<td>282 × 32</td>
<td>Bottleneck</td>
<td>6</td>
<td>64</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>12–14</td>
<td>142 × 64</td>
<td>Bottleneck</td>
<td>6</td>
<td>96</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>15–17</td>
<td>142 × 96</td>
<td>Bottleneck</td>
<td>6</td>
<td>160</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>72 × 160</td>
<td>Bottleneck</td>
<td>6</td>
<td>320</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>72 × 320</td>
<td>Conv2d 1 × 1</td>
<td>—</td>
<td>1280</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>72 × 1280</td>
<td>AvgPool 7 × 7</td>
<td>—</td>
<td>—</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>21</td>
<td>12 × 1280</td>
<td>Conv2d 1 × 1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>-</td>
</tr>
</tbody>
</table>
activation function to accelerate the network’s convergence and prevent overfitting during the training process.

3.2. Data Quantization. To simplify the inference process of the model and reduce the occupation of computing resources, this paper adopts compression optimization strategies such as BN layer fusion and dynamic-precision data quantization.

It is difficult for the BN layer and convolutional layer to share the hardware resources in the hardware deployment of the CNN model. Hence, we fuse them to generate a new convolution layer (the same size but different weights). The computations of convolution and BN are calculated as follows:

\[ y_{\text{conv}} = w \cdot x + b, \]
\[ y_{\text{bn}} = \gamma \frac{y_{\text{conv}} - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} + \beta, \]
\[ = \gamma \left( \frac{w \cdot x + b - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} \right) + \beta, \]

where \( w \) and \( b \) are the weight and bias of the convolution layer. \( E[x] \) and \( \text{Var}[x] \) are the mean and variance of the feature map, \( \gamma \) and \( \beta \) are the scaling factor and bias, and \( \epsilon \) is a small positive number to prevent the denominator from being 0. These parameters are obtained by the BN layer in the training stage and fixed in the inference phase. The new fused weights \( \bar{w} \), bias \( \bar{b} \), and the computations of the new fused layer can be expressed by equations (2) and (3).

\[ \bar{w} = \frac{\gamma}{\sqrt{\text{Var}[x] + \epsilon}} w, \]
\[ \bar{b} = \frac{\gamma}{\sqrt{\text{Var}[x] + \epsilon}} (b - E[x]) + \beta. \]

Based on the abovementioned operation, the BN layer is fully integrated into the convolution calculation in the inference stage without any loss of accuracy. This can effectively reduce computation and resource consumption while accelerating the reasoning process.

Previous works have shown that the 16-bit quantization guarantees almost no accuracy loss, while 8-bit quantization leads to more than 3% loss of accuracy in MobileNetV2 [48, 49]. Therefore, the paper refers to the literature [50, 51] and adopts a hardware-friendly quantization method to quantify the new convolution layer while maintaining accuracy.

The floating-point values and fixed-point values are represented as \( x_{\text{float}} \) and \( x_{\text{fixed}} \), respectively; their transformation relationship is formulated as follows:

\[ x_{\text{fixed}} = x_{\text{float}} \times 2^{Q}, \]

where \( Q \) is the quantization bit width. The optimal \( Q \) values of the weights, bias, input, and output in the network are determined by minimizing the errors before and after quantization.

The whole model has \( L \) convolution layers, and fixed-point quantization is used in each convolution layer, which includes fixed-point quantization of input \( x_{l} \), weight parameter \( W_{l} \), bias parameter \( b_{l} \), and output \( y_{l} \). The calculation equations are as follows:

\[ y_{l} = \sum_{i=1}^{L} W_{l} \cdot x_{l} + b_{l}, \]
\[ y_{l}' = 2^{-Q_{y}} y_{l} = 2^{-Q_{y}} w_{l} \cdot 2^{-Q_{w}} x_{l} \cdot 2^{-Q_{b}} + b_{l} \cdot 2^{-Q_{b}}, \]
\[ y_{l}'' = \sum_{i=1}^{L} w_{l} \cdot x_{l} \times 2^{-Q_{w}} \cdot x_{l} + b_{l} \cdot 2^{-Q_{b}} + b_{l}, \]

where \( Q_{x}, Q_{w}, Q_{b}, \) and \( Q_{y} \) are the quantization bit width of the input, weight, bias, and output, respectively, and \( x_{l}, w_{l}, b_{l}, \) and \( y_{l} \) are the quantized input, weight, bias, and output, respectively.

The multiplication and accumulation operation of floating-point numbers in equation (5) is converted to the multiplication and accumulation, and shift operation of fixed-point numbers by equation (7).

4. The FPGA-Based MobileNetV2 Accelerator

This section proposes a low-power MobileNetV2 accelerator based on FPGA for real-time seed sorting. The architecture and computing module are proposed.

4.1. Overall Architecture. This design is a low-power seed sorting accelerator that can implement high-precision sorting on a resource-limited FPGA platform. The architecture overview of our proposed MobileNetV2 accelerator is illustrated in Figure 2.

The architecture mainly consists of a processing system (PS) and programmable logic (PL). PS is responsible for the program scheduling and function configuration, while PL is responsible for calculating acceleration. Double data rate synchronous dynamic random access memory (DDR SDRAM) is the external memory, asynchronous response mode (ARM) is the system master control unit, and advanced extensible interface (AXI) bus is the on-chip bus for the communication between the processor and FPGA. The quantized model parameters are preloaded into DDR during the inference. Data are currently transferred to the PL through the AXI bus, which input them to the computing module to conduct the calculation. Almost all the operations of CNN are carried out in the computing module, containing a standard convolution (SC) computing engine, DWC computing engine, PWC computing engine, fully connected (FC) computing engine, data loading/storing module, and memory management unit (MMU). After the calculation, the results are returned to the on-chip cache as the resulting output or as the input of the following layer network. In this way, the whole model’s calculation is completed, and the final seed sorting result is obtained.
4.2. Block Convolution Strategy. Due to the limited on-chip storage, the middle layer results of the MobileNetV2 model are transferred back and forth between on-chip memory and off-chip memory, resulting in latency and energy consumption increases that cannot be ignored [52]. To address this issue, we design a block convolution strategy, a hardware-friendly and efficient convolution operation that can altogether avoid the off-chip transfer of intermediate feature maps. Block convolution is performed by splitting the feature map into independent blocks, where an individual block can be convolved separately, as shown in Figure 3. Converting a standard convolution with kernel size $K$, stride $S$, and padding size $P$ into a block convolution with the proper blocking number $N$ and block padding size $P_t$, the process can be explained by the following formula:

$$I + 2P - K + 1 = N \left( \left\lfloor \frac{I}{N} + 2P_t - K \right\rfloor + 1 \right).$$

(8)

where $I$ denotes the input feature maps. The relationship between $P_r$ and $P_{ri}$ can be determined by equation (9):

$$P_{ri} = S \times P_r + K - 2P_t.$$  

(9)

Figure 3(a) shows the block convolution calculation of the input fixed block $F_{in} \in R^{P_r \times P_r \times P_m}$ and convolution kernel fixed block $W \in R^{K \times K \times P_r \times P_m}$. The first fixed block of the input feature map is convolved with Split1, Split2, and Split3 of the corresponding weight position, and the intermediate result $F_{out} \in R^{P_r \times P_r \times P_m}$ is obtained. Then, calculating the second fixed block and the pixel value of each channel is added to the corresponding position of the intermediate result of the previous stage. Eventually, the complete results are obtained. Each input channel of standard convolution must be convolved by using a specific kernel, and then, the result is the sum of the convolution results of all channels. Differently, DWC is performed for each input channel separately, and the feature map channels remain unchanged before and after convolution. In the spatial dimension, the complete feature map depth-wise convolution is divided into many independent subfeature map depth-wise convolution, as shown in Figure 3(b). Considering data reuse and
improving memory access efficiency, PWC first converts the input feature map and weights flattening into matrices through \( \text{img2col} \) conversion operation and then reduces the number of data access through the block convolution strategy that decomposes the matrix multiplication with large dimensions into multiple small-matrix multiplications, as shown in Figure 3(c). The \( T_p \times P_n \) input matrix and \( P_m \times P_n \) weight matrix can be calculated simultaneously in each cycle to speed up matrix calculation. Block convolution is essentially splitting the whole feature map into multiple sub-feature maps into spatial dimensions. Each sub-feature map is then convolved separately, and the results are stitched together.

In addition, some optimization techniques are commonly also adopted, including loop tilling, loop unrolling, loop pipelining, and loop transformations.

4.3. Memory System Design. An efficient memory system should be able to standardize data access mode according to the channel parallel mode of block convolution. In this section, the design of the storage system is introduced, mainly including the creation of a memory management unit (MMU) and the method of data buffering.

4.3.1. Memory Management Unit. The traditional row-wise data arrangement first arranges data along the row direction and then along the column and channel directions in turn, which is only applicable to the convolution mode of channel separation, as shown in Figure 4(a). Block convolution performs convolution on each fixed block in the spatial dimension. If row-priority data arrangement is still used, it will bring difficulties in data acquisition and slow down the computational efficiency due to the discontinuity of input. To solve these problems, we optimize the data arrangement mode in the storage space and propose a novel data reordering mode, as shown in Figure 4(b). Since the data are first arranged along the channel direction, it can be directly superimposed to obtain the output result without consuming the cache to store the intermediate result of the operation. This format dramatically reduces unnecessary access delays to external memory, resulting in high-peak performance.

4.3.2. Buffer Design. In order to continuously provide data for computing engines at each layer, the data double-buffering mechanism is adopted, as shown in Figure 5(a). When
the input buffer A data are transferred to the computing engine for calculation, the next batch of data are simultaneously loaded from off-chip memory into input buffer B. Next, the input buffer B data are transferred to the computing engine, while the input data are loaded from the off-chip memory into the input buffer A. This double-buffering scheme ensures that the data computing unit is always working during the whole process. The data loading and data computing unit are completely parallel operations, which effectively alleviate the memory access bottleneck problem so that the computing efficiency can be improved significantly.

4.4. Computing Engine Design. To reduce the redundancy of data access and improve the parallelism of computation, we design two computing engines to accelerate convolution according to different convolution modes; one is responsible for DWC, as shown in Figure 5(b), and the other is responsible for PWC, as shown in Figure 5(c).

4.4.1. DWC Computing Engine. DWC is to filter each feature image in the input channel and extract the shape, contour, and other complex features of the seed image, which has the same channel number between input and output. The computing engine consists of a multiplier array, adder tree, nonlinear activation ReLU6, and controller modules, and it conducts the computation of the whole model. The computing engine is equipped with a set of processing elements (PEs), each PE is responsible for the 2-D convolution calculation of a single input feature map with the corresponding weight, and multiple PEs constitute a DWC in parallel. A buffer is used behind each PE each PEs cache data that will be reused shortly. This buffer prevents these data from being dropped and avoids redundant memory operations that reload these data from off-chip memory.

This computing engine can also be configured to perform SC. SC is only adopted for the first convolution level to avoid too much information loss. For the seed sorting task, the channel number of the input feature map is 3. Therefore, the computing engine performs block convolution under the condition that the input feature mapping channel $P_n$ is 3. In addition, PE with different parallel factors $P_n$ and $P_m$ can be designed according to different input channels and parameters of varying convolution layers.

4.4.2. PWC Computing Engine. PWC of the extension layer improves the dimension of the input feature channel to retain more detailed texture information such as edge, corner, and color of seeds. In this paper, the GEMM is used...
to realize the PWC operation; the computing engine is shown in Figure 5(c). First, before the input feature maps are fed to PWC computing engines, they need to perform an \textit{img2col} conversion operation. Then, the kernel and feature map are input into the PE of the GEMM algorithm, where high-speed parallel computation is conducted through a multiplier array and addition tree. Finally, the calculation results of PE and bias are accumulated through the accumulator. The output is stored in the output buffer as the intermediate result or the input of the next DWC layer. The input feature map is continuously arranged in memory after being converted to a matrix by the \textit{Im2col} operation. Therefore, GEMM, a fast convolution method of matrix multiplication, is adopted to continuously access the input feature map, dramatically improving access efficiency.

In order to make full use of all multipliers in the acceleration engine, as shown in Figure 5(d), the matrix multiplication with large dimensions is decomposed into multiple small-scale matrix multiplications, and the input feature map is divided into several submatrices, which are successively moved into the buffer of the acceleration engine in Figure 5(a). Finally, small-scale matrix multiplication results are added to the PE array. For an acceleration computing engine, multiplications can be performed simultaneously.

The fully connected layer consists of the global average pooling and the PWC layer, so the computing engine in Figure 5(c) is also suitable for the fully connected layer.

5. Experimental Results

This section describes the experimental setup, the evaluation performed, and an analysis of the results. All performance and execution time measurements are from a natural experimental system.

5.1. Experimental Dataset. To verify the effectiveness of the seed sorting method based on FPGA, maize seed dataset and red kidney bean dataset are selected for the experiment.

The maize seed dataset [24] is a haploid and diploid corn seed dataset published by Sakaria Maize Research Institute in Turkey, including 3,000 RGB images of Maize seeds. Of which 1,230 are haploid seed images, and the rest 1,770 are diploid seed images. 861 haploid images and 1239 diploid images are selected as the training set, and the rest are used for testing. Figure 6(a) shows some typical samples of the dataset.

Red kidney bean dataset. In order to verify the effectiveness of the accelerator based on FPGA in a multi-classification seed sorting task, a red kidney bean seed dataset was built. The acquisition equipment is a 1/2.5 CMOS camera, which uses a white circular light source to supplement light and a white background plate to distinguish red kidney beans from the background more accessible. A total of 3,831 samples are collected, and red kidney
beans are divided into plump beans (1,661), peeled beans (509), dried beans (1,173), and broken beans (488) according to the quality evaluation grading standards adopted by the enterprise. Some sample images from the dataset are shown in Figure 6(b).

5.2. Experimental Settings

5.2.1. Model Training. Model training was carried out on Ubuntu 18.04 and the deep learning software framework is PyTorch. All training evaluations and measurements are performed on an NVIDIA 1080 TI GPU, with CUDA10.1 accelerating calculation. Referring to the network parameter configuration of the MobileNetV2 classification model [15], we train the network by using small-batch stochastic gradient descent (SGD), with an initial learning rate of 0.01, and the learning rate is adjusted to 1/10 of the original after 50 epochs. The number of epochs for training is 150, the batch size is set as 32, the momentum parameter is set as 0.9, and the weight decay parameter is set as 0.0001.

5.2.2. Model Testing. In the experiment, Vivado HLS 2019.2 was used for accelerator logical IP design, and Vivado 2019.2 was used for running simulation and synthesis, reporting power, and generating bitstream. After that, the application program development and debugging of the seed sorting algorithm are carried out in Vitis. Finally, we choose the Xilinx XC7Z020 as our target platform to verify the proposed MobileNetV2 accelerator for real-time seed sorting, as shown in Figure 6(c).

Table 3: Performance of some popular CNN in red kidney bean and maize test sets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameters (M)</th>
<th>Macs</th>
<th>Acc (red kidney bean) (%)</th>
<th>Acc (maize) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet18</td>
<td>11.18</td>
<td>1.82 G</td>
<td>96.72</td>
<td>94.48</td>
</tr>
<tr>
<td>DenseNet</td>
<td>6.96</td>
<td>2.88 G</td>
<td>96.07</td>
<td>94.65</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>9.94</td>
<td>1.51 G</td>
<td>95.94</td>
<td>94.30</td>
</tr>
<tr>
<td>ShuffleNet 1.5x</td>
<td>2.48</td>
<td>301.90 M</td>
<td>96.85</td>
<td>94.48</td>
</tr>
<tr>
<td>MixNet-m</td>
<td>3.48</td>
<td>337.64 M</td>
<td>96.59</td>
<td>94.65</td>
</tr>
<tr>
<td>MobileNetv2 (Ours)</td>
<td>2.23</td>
<td>320.24 M</td>
<td><strong>97.91</strong></td>
<td><strong>96.50</strong></td>
</tr>
</tbody>
</table>

Table 4: Accuracy on different platforms.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Accuracy of PC (%)</th>
<th>Accuracy of FPGA (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red kidney bean</td>
<td>97.91</td>
<td>97.42</td>
</tr>
<tr>
<td>Maize seed</td>
<td>96.50</td>
<td>95.60</td>
</tr>
</tbody>
</table>

Figure 7: The comparison of raw images and grad-CAM visualization of seeds on the MobileNetv2 network (a), (c) raw images; and (b), (d) Grad-CAM visualization.
superior to the state-of-the-art methods. We also obtain the highest accuracy of 96.50% on maize seed.

To better explain the seed sorting effect of the MobileNetV2 network, we visualize the important feature areas that the network focused on by using Grad CAM. Figure 7 shows the visualization results of the MobileNetV2 model on maize and red kidney bean seed images. The network selectively emphasizes the characteristics of seed image information, thus improving performance. For example, Figure 7(b) shows visualizations of haploid and diploid maize seeds. The model focus on the kernel region of the seed, where the two types of seeds are different. Figure 7(d) shows the visualized results of red kidney beans of different varieties. The model gives maximum attention to each damaged area of the seed, which is consistent with human perception.

After the network of the seed sorting accelerator is determined, the software and hardware systems are built based on the Xilinx ZC7020 development board, and the MobileNetV2 acceleration system with 16-bit fixed-point accuracy is realized. The test accuracy of seed classification on different platforms is shown in Table 4. The accuracy of PC is the recognition accuracy before quantization. After quantization, the forward network inference is carried out on FPGA. It is found that the recognition accuracy of red kidney bean and maize seed after quantization is only reduced by 0.5% and 0.9% compared with that before. Experimental results show that the quantization method adopted in this paper has little influence on the accuracy of the model and can be ignored.

The seed sorting hardware accelerator designed in this paper can recognize more than 95% of two kinds of different seeds, indicating that the sorting performance of the system is outstanding. Therefore, we chose MobileNetV2 to make it easy to deploy on edge devices to achieve fast and high-purity seed sorting.

### 5.3.2 Design Space Exploration

The configurable parameters in the CNN accelerator determine the parallelism and throughput of the system. However, it is challenging to balance resources and performance when deploying CNN accelerators on FPGA with limited resources.

We refer to the space exploration approach in the literature [36] to help design the accelerator by using the roofline model and analyze its performance. Combining the computation to communication (CTC) ratio and physical compute space and analyzing the relationship between the on-chip cache and the roofline model of the ZYNQ 7020 platform, we seek the optimal design and explore the parallelism configuration under each layer of block convolution. The mathematical relationship between the blocking factor and the CTC ratio is shown in equation (10), where $\beta_{in} = P_m \times P_n \times P_{i\alpha}$, $\beta_{out} = P_m \times P_n \times P_{c\alpha}$, $\lambda_{in} = \lambda_{out} = R \times C \times M \times N / P_m P_n P_c$, and $\lambda_{out} = R \times C \times M / P_m^2 P_c$, $R, C, M$, and $N$ are the width and height of the output feature map, the number of output channels, and the number of input channels, respectively.

$$\text{CTC Ratio} = \frac{\text{number of operations}}{\text{number of external access bytes}} = \frac{2 \times R \times C \times M \times N \times K \times K}{4 \times (\lambda_{in} \times \beta_{in} + \lambda_{out} \times \beta_{out})}$$

(10)

The computational performance can be calculated using equation (11).

$$\text{computational performance} = \frac{\text{total number of operations}}{\text{number of execution cycles}} \approx \frac{2 \times R \times C \times M \times N \times K \times K}{M/P_m \times N/P_n \times R \times C \times K \times K}$$

$$\approx 2P_m P_n$$

(11)

The on-chip buffer resources are strongly related to the blocking factor, as shown in equation (12).

$$\text{on-chip resources} = \text{Input Buffer} + \text{Weight Buffer} + \text{Output Buffer} = \frac{P_m \times P_n}{2} + \frac{P_m \times P_n \times P_{i\alpha}}{224} + \frac{P_m \times P_n \times P_{c\alpha}}{224}.$$

(12)

The optimization objective of space exploration is throughput performance. On-chip DSP, BRAM, and DDR bandwidth resources are the constraints for the whole design space exploration, as shown in equation (13). The optimal parameter configuration [24, 128, 64] is obtained through design space exploration, which can achieve 10.86 GOPS computing performance.

$$\begin{align*}
\text{performance} \leq \min \{\text{computational performance}, \text{CTC Ratio} \times BW\}, \\
\text{on-chip cache resources} \leq \text{BRAM total} = 140, \\
s.t.: & \quad P_n \times P_m \leq \text{DSP total} = 220, \\
& \quad 0 \leq P_n \leq N, \\
& \quad 0 \leq P_m \leq M, \\
& \quad 0 \leq P_{i\alpha} \leq R, \\
& \quad 0 \leq P_{c\alpha} \leq C. 
\end{align*}$$

(13)

Figure 8 shows the resource utilization and throughput performance for some representative configurations. When accelerator parallelism is low, BRAM dominates accelerator performance. BRAM is mainly used to implement input cache and output cache. As the degree of parallelism increases, the use of DSP increases significantly, and DSP becomes the primary resource constraint. The computational performance of the accelerator increases with the increase of parallelism. Since the PWC layer accounts for the most significant proportion in MobileNetV2, when resource utilization approaches the upper limit, more resources are allocated to the PWC layer to achieve greater parallelism by sacrificing the resource usage of the SC layer.
The detailed resource utilization is shown in Table 5. The utilization rate of DSP and BRAM is exceptionally high. A large amount of DSP resources is consumed because input-output parallelism under the block convolution strategy requires many parallel multiply accumulation and shift calculations. BRAM utilization is up to 91%, and almost all BRAM resources are used. The high utilization of BRAM reduces the data access to the off-chip memory, indicating that the input/output double buffering adopted in this paper increases the consumption of BRAM on-chip storage resources and achieves the purpose of access optimization by increasing hardware resources. The proposed architecture...
generally achieves better performance while optimizing resource allocation on-chip, which is a feasible solution for deploying embedded CNN.

5.3.3. Comparison with Previous FPGA Accelerators. Previous studies [56] have shown that an essential limitation of FPGA-based CNN accelerators is that the highest performance achieved depends entirely on the hardware structure of the processor, that is, the number of DSPs (MAC units) used, leading to a natural advantage in designing based on resource-rich hardware platforms. To make a fair comparison across different platforms, in addition to computational performance GOPS, the throughput achieved per DSP block (GOPS/DSP) is employed to evaluate DSP computational efficiency.

This paper constructively proposes the block convolution strategy, a low-density FPGA-friendly approach to avoid the increased latency and energy consumption caused by large-scale intermediate result off-chip data transfers. An efficient storage system with a double buffering mechanism and a new data reordering model is proposed to design a fantastic acceleration architecture. Table 6 provides comparisons with previous accelerators in terms of GOPS and GOPS/DSP metrics. Most of these FPGA accelerators are explicitly designed for a particular network or networks with similar model sizes in one prescribed task. Because of the proposed flexible and reconfigurable architecture, our accelerator can support more types of networks with comparable or better performance. Our implementation achieves 10.86 GOP/s performance, which shows 2.68× and 2.82× speedup compared with references [59, 60]. However, compared with FPGA platforms [36, 45, 57] with richer hardware resources, higher integration, and more advanced architecture, there are still some gaps. Remarkably, as illustrated in the last line of Table 6, our method achieves the best DSP computational efficiency, which provides 0.92× to 5.70× speed up compared with previous dense accelerators.

Table 7 shows the real-time performance compared with the previous works. Our implementation can achieve 29.4 FPS on MobileNetV2 with an architecture configuration of 0.75, which fully meet the real-time and precision stability of seed sorting. This accelerator is proven to be highly scalable to accommodate FPGAs of different capacities and CNNs with various parameter configurations. Table 8 shows a direct comparison with previous CNN accelerators in terms of the FPS/DSP metrics. These works all implement the MobileNetV2 accelerator architecture based on FPGAs. Reference [48] uses the Intel® Arria®10 as the target platform to achieve the highest FPS/DSP at a high cost. Our design achieves the second-highest FPS/DSP on a low-density FPGA, far outperforming low-cost accelerators.

Compared with existing methods, the proposed design in this paper achieves significant improvement in both DSP computation efficiency and logic resource utilization. Even though our target device is substantially resource-constrained than the ones utilized in the earlier studies mentioned above, we are still able to achieve competitive FPS levels that can satisfy practical industrial needs. The power consumption of the seed sorting system is only 3.01 W. Compared with the high-power GPU cluster, the seed sorting system based on FPGA is easier to meet the low-power requirements of the mobile terminal and increase the service time of the mobile device. The work in this paper combines the characteristics of low power consumption and high computing performance, which has a particular reference value.

6. Conclusion

In this work, a flexible and hardware-parallelized MobileNetV2 accelerator is proposed to aim at the requirement of rapid sorting for high-resolution seed images. We design from three perspectives: dynamic point quantization, parallel computing, and efficient memory system design. The proposed approach is implemented on the resource-limited
ZYNO-7020 FPGA platform for the seed image classification task, which achieves 29.4 FPS and 0.92x to 5.70x DSP-efficiency over previous CNN accelerators. Experiments show that the accelerator designed in this paper can carry out high-precision real-time sorting of different types of seeds, which provides new ideas for deep learning models to be transplanted into automatic agricultural seed sorting equipment.

The current work is still mainly focused on the recognition algorithm and the processing of static images, and the real-time sorting device under dynamic is not yet perfect. For future work, a highly automated sorting device with both automatic identification and automated sorting will be implemented to achieve high-speed and accurate sorting of seeds.

Data Availability
The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest
The authors declare that they have no conflicts of interest.

Acknowledgments
This work was supported by the NSFC (No. 62072489, U1804157), Henan Science and Technology Innovation Team (CXTD2017091), IRIS-TSHN (21IRIS-TSHN013), the Zhongyuan Science and Technology Innovation Leading Talent Program (214200510013), and Qianjiang laboratory open fund project of the Hangzhou Research Institute of Beihang (2020-Y3-A-026).

References


