Received 5 May 2023, accepted 16 May 2023, date of publication 19 May 2023, date of current version 1 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3277857

RESEARCH ARTICLE

Hardware Architecture Design for Hand Gesture Recognition System on FPGA

TSUNG-HAN TSAI, (Senior Member, IEEE), YUAN-CHEN HO, AND PO-TING CHI
Department of Electrical Engineering, National Central University, Zhongli, Taoyuan 320317, Taiwan

Corresponding author: Tsung-Han Tsai (han@ee.ncu.edu.tw)

This work was supported by the Ministry of Science and Technology, Taiwan, under Grant MOST 111-2221-E-008-089-MY3.

ABSTRACT Hand gesture recognition (HGR) is one of the widely-used human-computer interaction technology. With HGR, the user can operate the interaction system without touching any devices. For a better experience, recognition accuracy and computational speed are especially important. In this work, a small-footprint HGR model and its hardware architecture design are proposed. The model first processes the hand segmentation and uses the feature to recognize the hand gesture. The model mainly consists of depthwise separable convolution to reduce the overall parameters and computations. We transfer the hand segmentation task with some features to the hand gesture recognition task as a single-stage model. Based on this hardware-efficient model, we propose the hardware architecture of the whole neural model including depthwise convolution, pointwise convolution, batch normalization, and max-pooling. We also demonstrate it on the evaluation board. The whole system is implemented on the Xilinx ZCU106 evaluation board. The implemented system can achieve the performance of 52.6 fps and 65.6 GOPS based on the evaluation board.

INDEX TERMS Hand gesture recognition, attention model, depthwise separable convolution, hardware accelerator, field-programmable gate array (FPGA).

I. INTRODUCTION

Human-computer interaction is widely used in recent years. Typically, voice and hand gestures can be easily used to control the target system without touching devices. Hand gesture recognition (HGR) can operate the system more intuitively and precisely. For example, it can be used for smart TV and video games. Currently, most HGR work can be divided into two categories according to the form of input. One is using wearable devices, such as data gloves, bracelets, etc. These devices are inconvenient and limited by the environment [1]. The other is using image sensing [2]. By using an image as the input, the user can operate the system unrestrictedly without wearing any devices.

However, since less information can be determined by the gesture, some research exploits depth cameras (such as Kinect, RealSense) [3] or two cameras for stereo matching to obtain the depth information. Although it improves the recognition accuracy in a severe environment such as complex scenes, the depth cameras are relatively bulky and costly, and also have certain bottlenecks to apply to smart home appliances. Compared with the image with depth information, the HGR using a single RGB image is easier to apply to various scenes and also reduces the cost. However, it needs more algorithm effort, which may require computational resources and increase the recognition latency. Therefore, it is necessary to optimize the HGR process and implement the system for real-time recognition with good accuracy.

Recently deep learning network has reached great success in several kinds of computer vision tasks, including object classification [4], object detection [5], and semantic segmentation [6]. These successes drive the development of HGR systems based on deep learning methods [7], [8]. It has been proven that processing the hand segmentation before the hand recognition improves overall performance. The segment feature filters out the complex image background, differences in different skin colors, shadows, and other lighting changes. Then the recognition network concentrates on gesture classification. By doing so, the recognition has significant improvement, especially in the complex scenes.

The associate editor coordinating the review of this manuscript and approving it for publication was Ilaria De Munari.

In neural network computation, the computational latency and memory usage are mainly affected by the model size. Currently, most HGR models include massive parameters or complex data paths. For example, in [9], the HGR-Net exploits dual channels which are the original feature and the segmented feature to improve accuracy. Although it can achieve good recognition results, memory usage cannot be efficiently reduced because of the temporary data generated from different channels.

Therefore, we designed a small-footprint model called HGR-Lite as a hardware-efficient model. The proposed model size is only 1.07MB, which is less than half of HGR-Net. The data path of the proposed model is straightforward, *i.e.* there is no extra temporary data should be saved. Additionally, this model mainly consists of depthwise separable convolutions [10], which perform fewer computations and are more efficient than conventional convolutions.

This paper focuses on the architecture design of the proposed HGR-Lite model. Since the trend is to implement the neural network model on the embedded system, we consider it and design the hardware accelerator for the deep neural network (DNN). Although CPUs and GPUs are more flexible and convenient for neural network training and development, they are not suitable for the inferencing stage in home appliances because of their high power consumption and cost. We use FPGAs to design since they are more suitable than CPUs and GPUs in terms of performance and power.

We use the dedicated hardware architecture and, implement this end-to-end HGR system on FPGA. The neural network is composed of many processing elements (PEs) for parallel computations. Since most computations concentrate on depthwise separable convolution, we focus on optimizing the depthwise convolution and pointwise convolution respectively. Also, we exploit the ping-pong buffer to increase the memory bandwidth for real-time recognition. After all the processes, the result is shown on the monitor through the HDMI.

In summary, our system has better performance in terms of less training and computing costs in both indoor and outdoor scenes. The whole system is implemented on FPGA to verify its performance. The proposed HGR system has the following key contributions.

1. We design a small-footprint model, HGR-Lite, for the HGR task. It can recognize 11 different gestures including the unknown. This model is also greatly reduced by the parameters and calculations while maintaining recognition accuracy.
2. We design the pure-hardware end-to-end gesture recognition system on ZCU106 FPGA without any support from the CPU. The system includes all the necessary sub-modules such as YUV to RGB module and HDMI module, making the system more comprehensive.
3. A memory-efficient neural network accelerator is proposed for improving the memory bandwidth issue between the depthwise convolution and pointwise convolution. Also, the pipelined architecture is derived to achieve real-time processing.

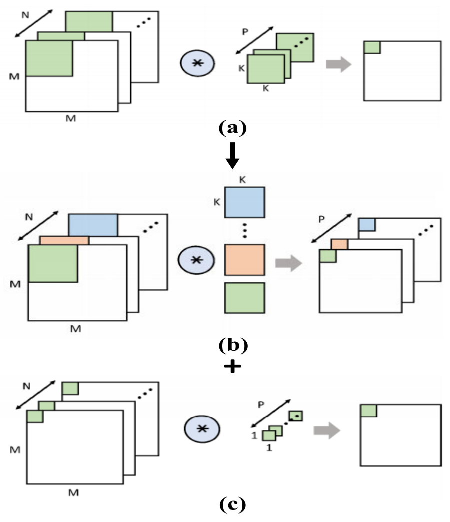

FIGURE 1. (a) standard convolution. (b) depthwise convolution. (c) pointwise convolution.

The remainder of this paper is organized as follows. Section II discusses the related works of hand segmentation, hand gesture recognition, and DNN hardware accelerators. Section III describes the design of our proposed HGR-Lite model and the result. Section IV gives the DNN accelerator architecture of our proposed system. Section V presents the experiment results, and Section VI concludes this paper.

II. RELATED WORKS

A. HAND GESTURE RECOGNITION

In the HGR algorithm, many kinds of research use the feature extraction algorithm to obtain the features of the hand region in the image and then use the classifier to classify the gesture. Reference [11] used the Bayesian model to detect hand regions in an image. Hand features of low-level (color) and high-level (shape, texture) are used in gesture recognition and classified by SVM classifier. However, due to the high complexity of the algorithm, the computation time is large. Also, the dataset used in their work is limited to a specific background and cannot be widely used in all scenarios.

Depth cameras have also been applied to hand gesture recognition [12], [13]. In [12], the authors used the Leap Motion and Kinect devices to combine depth information and skeleton joint descriptors. Then SVM classifier with a Gaussian Radial Basis Function (RBF) kernel is used to classify the gesture. In [13], the k-curvature algorithm is used to describe the posture by extracting the contour of the hand. Then a dynamic time-variant algorithm is used to identify and compare it with a series of pre-recorded gesture labels. This technique is used for the classification of static and dynamic gestures. Although these works may have good recognition accuracy, the depth camera is relatively costly compared to a single CMOS camera.

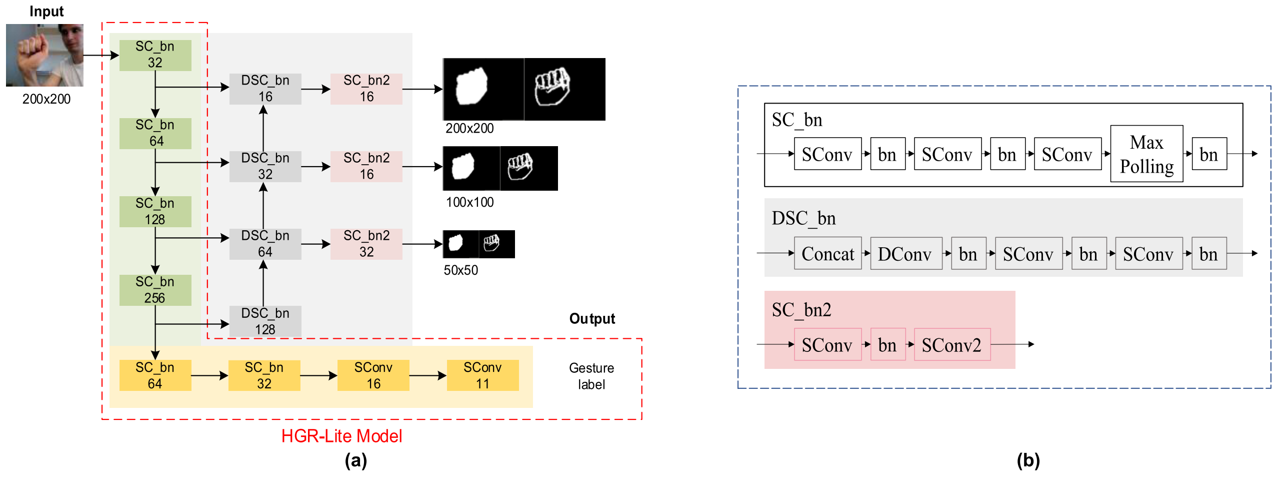

FIGURE 2. **HGR-Lite model. (a) main model. (b) layer structure in model.**

With the advance in deep learning, [14] proposed a multi-channel CNN for HGR. It uses the input image to calculate the grayscale image, and edge feature image of horizontal and vertical and then uses a simple network model to calculate the classification results. Reference [15] proposes an architecture for automatic gesture recognition that globalizes local features to improve performance. Reference [16] designed a new deep learning neural network model integrating several advanced action recognition techniques to solve the complexity and performance problems of dynamic gesture recognition. Although the network architecture in [17] is simple, it uses Sobel filters to calculate edge features. The dataset used in their work is almost in the simple background, making it difficult to apply to real scenes.

In [9], the HGR-Net model was proposed and widely applied in hand gesture recognition. The input image is calculated by the hand segmentation sub-model, and then the recognition sub-model is used to classify the hand gesture. Such a model can train the hand segmentation sub-model first. When the segmentation sub-model is trained, the recognition sub-model is added to co-training to improve the recognition rate of the gesture. Reference [9] was trained using the OUHANDS dataset [17]. Compared to the other large model-size neural networks (ResNet-50: 99MB, MobileNet: 16MB), HGR-Net has an average recognition rate of 88% at a model size of 2.4MB. However, the data flow in HGR-Net is relatively complex. The dual stream dataflow needs more memory usage to store the temporary data. Besides, the used parameters are large and could be further reduced.

B. HARDWARE ACCELERATOR

In recent years, the design of neural network accelerators has been a research hotspot in the field of computer vision. DNN or CNN is implemented on CPU, GPU, and even on FPGA and ASIC [18]. There are three challenges in the design of hardware-based neural network accelerators. One is the mapping of multiple features in a convolutional layer with large weights. It induces a large storage size. Second, the high complexity of convolution calculations largely increases the required time for the inference process. The third is the data access delay caused by the limited memory bandwidth. These factors hinder the real-time performance and widespread deployment of DNN especially in resource-constrained embedded systems. Therefore, it is necessary to consider the speed and memory balance to achieve optimized hardware performance.

In the research of hardware accelerators, [19] proposed the CNN accelerator designed on Xilinx Vertex 7 FPGA. The most important issue in implementing a neural network on FPGA is that the calculated throughput and memory bandwidth may not match. Due to insufficient utilization of hardware resources or memory bandwidth, the existing designs cannot achieve optimal performance. Therefore [19] proposed an analysis design using the roofline model, and used various optimization techniques such as cycle scheduling and pacing. It analyzes the calculated throughput and the required memory bandwidth and then uses the roofline model to determine the best performance with low hardware resources.

In 2018, [20] proposed Angel-Eye's hardware accelerator design. It is a programmable and flexible CNN accelerator architecture to explore the hardware design process. This process obtains weight parameters and network topology from the trained model, then quantizes and compiles the parameter, and maps topology and quantized parameter to hardware.

Since CNN is a computationally intensive model, large amounts of computing resources are required in the inference stage. Therefore, in the hardware accelerator design of the FPGA, the trade-off between the number of calculations and the bandwidth of the memory is very important. To reduce the computational load of standard convolution operations, [21] proposed a depthwise separable convolution by splitting the standard convolution into depthwise convolution and pointwise convolution, and its calculation is shown in Figure 1., which has been applied in MobileNetV1 and later MobileNetV2. It is performed with

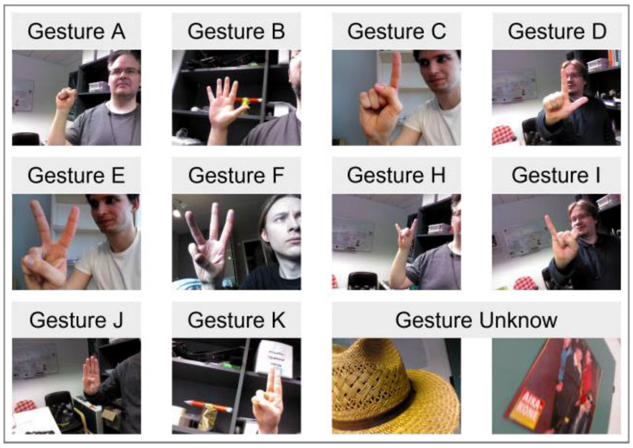

FIGURE 3. Classification of hand gesture.

fewer multiply-accumulate operations and weight parameters to achieve similar results from standard convolution. Based on the depthwise separable convolution, [22], [23] proposed the corresponding CNN accelerator. It is superior to the standard convolution accelerator design for both memory and computational speed.

III. NETWORK MODEL DESIGN

In our design, we establish a segmentation model to minimize memory usage. The hand segmentation task can share some features well with the hand gesture recognition task, making our design a single-stage model using the auxiliary task approach.

A. DESIGN OF HGR-LITE MODEL

In this work, we dedicate ourselves to implementing our work for real-world applications. Thus, the consideration of background diversity is especially important. We use PASCAL VOC2012 [24] as the background and hands data in OUHANDS to generate more data. OUHANDS contains 10 different gestures from 23 themes, divided into training, validation, and test datasets of 1200, 400, and 1000 images respectively. We refer to the concept of ResNet and FCN to design the hand-cutting sub-model with depth separable convolution and develop it in the framework of Tensorflow. Since the image resolution of the training dataset is different, we resize the image resolution to 200 $\times$ 200 pixels, which reduces the memory load of future hardware designs.

In the HGR network, we follow HGR-Net [9] to design the segmentation sub-model and the recognition sub-model to integrate the HGR model. In HGR-Net, there are two processing streams. One stream is to generate a hand segmentation map by the residual network. The other stream is to compute the RGB image. Finally, these two features are combined by the fusion function. Although the recognition accuracy of this design is largely increased, this two-stream model highly increases the computational complexity. Also, the second stream needs to stall until the hand segmentation map is computed.

TABLE 1. Segment evaluation of each training method.

	OUHANDS dataset + Pascal VOC 2012 as negative sample		
	Precision	Recall	F1
Baseline model	0.713	0.822	0.763
+data enhancement	0.953	0.887	0.919
+iteration training	0.992	0.939	0.965

The proposed HGR-Lite is shown in Figure 2(a). We dedicate ourselves to designing a hardware-efficient model with only a computation stream. The orange area is the new recognition sub-model. We use the pre-trained model of segmentation to assist the recognition sub-model for training. When performing the HGR task, we only need to use the model of the green and orange regions instead of using the entire model. These two modules are grouped as HGR-Lite mode in Figure 2(a). This design is to allow part of the hand segmentation sub-model (only green region when inferencing HGR task) as an attention model to learn hand features first and then help the recognition sub-model more effectively to perform the whole HGR task.

B. LAYER STRUCTURE IN HGR-LITE MODEL

We will discuss the layer structure of HGR-Lite in Figure 2(b), including the SC_bn module and DSC_bn module. Each SC_bn module includes three layers of depthwise separable layers, three layers of batch-normalization (BN) layers, and one layer of max-pooling layers. The number represents the size of convolution kernels. For example, 32 means the convolutional layer uses 32 convolution kernels to extract features. The output resolution of each SC_bn is 1/4 of the input. Each DSC_bn module contains one layer of deconvolution layer and two layers of depthwise separable convolution layer. Thus, the resolution of the output is four times the input.

After the DSC_bn module, two layers of depthwise separable layers are connected to output the hand segmentation and hand contours of different dimensions through FCN. By using the concept of ResNet, the feature images of the same resolution are combined and used as the input of the next layer. For example, the output of SC_bn32 will be merged with the output of DSC_bn32 as the input of DSC_bn16.

At the end of each training session, we verify the training dataset and copy the results including errors to a new folder of training data. We also fine-tune the model by taking the previously trained model as the pre-train model. After fine-tuning, we verify the training dataset again to increase the probability of selecting the error detection sample and then repeat this action until the accuracy of the training data no longer rises. We call it iterative training.

In the hand segmentation and recognition sub-models, we use a training strategy that includes a synthetic dataset combined with iterative training to help train a robust network model. The precision, recall, and F1 score are calculated. Since OUHANDS does not have background images

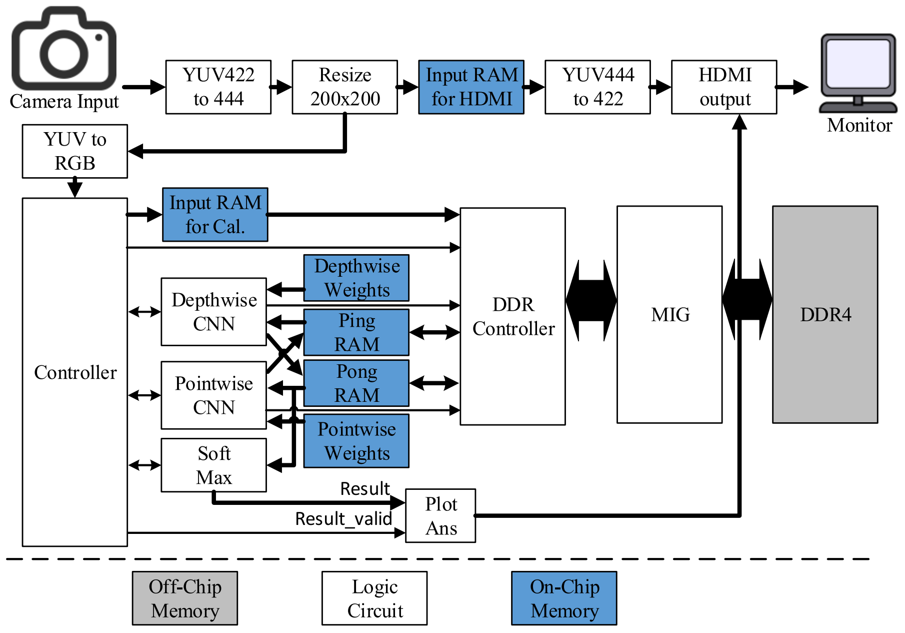

FIGURE 4. Block diagram of overall system.

as negative samples, we select 1000 images in PASCAL VOC 2012 as the negative sample and the OUHANDS test set as the positive sample to evaluate the overall performance.

As a result, the evaluation is shown in Table 1. The model trained without the enhanced data has a strong behavior of detecting the color of the skin. Thus, the sample with insufficient or too strong light cannot be accurately detected.

In the selection of training datasets, because only OUHANDS contains gesture labels, we use this dataset to train the recognition sub-model and fine-tune the part of the segmentation sub-model. At the same time, there are only 10 gestures in the OUHANDS dataset. We will mark the other datasets and negative samples beyond these 10 gestures as the 11th unknown label to increase the robustness of HGR, as shown in Figure 3.

IV. DESIGN OF HARDWARE ACCELERATOR

This section explains the design of the DNN hardware accelerator. To implement the HGR task in FPGA, we are faced with the problem of insufficient on-chip memory. Therefore, we analyze the bandwidth of the off-chip memory and the size limit of the on-chip memory and planned the best memory access method to reduce the access time with the off-chip memory. As a result, it helps the system to achieve the real-time HGR task.

A. OVERALL SYSTEM

We use the Xilinx ZCU106 evaluation board for development. It is the SOC architecture with the FPGA UltraScale+ MPSOC XCZU7EV. Note that our design does not use Processing System (PS) for control. All modules are implemented in Programmable Logic (PL).

The block diagram of the whole system is shown in Figure 4. The white block is logic circuits, the blue block is on-chip memory, and the gray block is off-chip memory. The image input is an SDI camera, and the image is transmitted to the FPGA through the FMC-SDI daughter board. Because the image format is YUV422, after capturing the image synchronization signal, the YUV422 is converted to the YUV444 first. Thus the image can be reduced to 200 $\times$ 200 pixels to match the input size of the HGR-Lite model.

From the 'Resize' module, the image will be divided into two ways. One is to store the image in 'Input RAM for HDMI' on-chip memory and then convert it to YUV422 format and output it to the screen through the HDMI interface. The other way is to convert the image to RGB for neural network operation where the RGB image will be stored in the 'Input RAM for cal.' through the 'Controller'. Then the image will be transferred to the 'DDR4' by the 'DDR Controller', and the image will be stacked to the parallel format for the usage of the hardware accelerator. At this time, 'Input RAM for cal.' will continue to store new input images, but it will stop transferring images to 'DDR4'.

When the input image is ready, 'Controller' will enable the 'Depthwise CNN' to start the depthwise convolution. In the beginning, the 'DDR Controller' will read part of the image from 'DDR4' and write it to the 'Ping RAM'. 'Depthwise CNN' will read the desired image from the 'Ping RAM' and start the operation. The data format of the 'Ping RAM' is to stack images of multiple channels on the same address for multi-channel parallelization convolution operation. The weight parameter will be used to shorten the parameter of the 32-bit floating-point parameter to the 16-bit fixed point, and then store the weight parameter of each layer in each

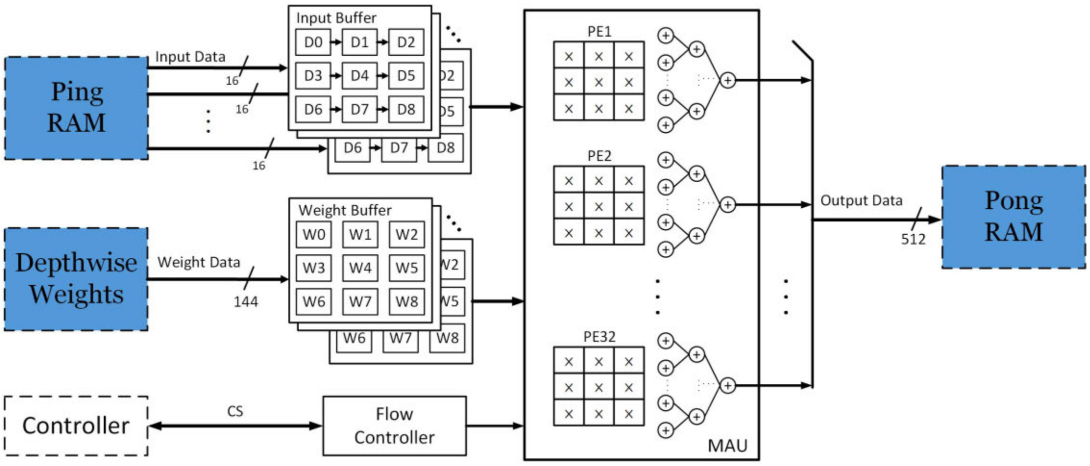

FIGURE 5. Architecture of depthwise CNN.

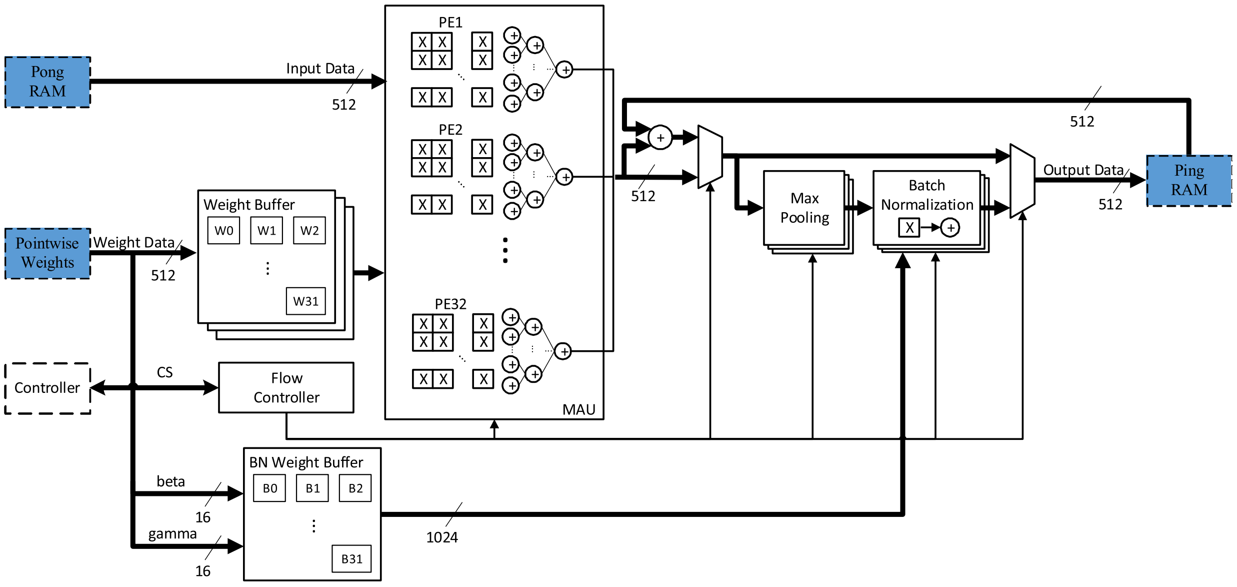

FIGURE 6. Architecture of pointwise CNN.

read-only memory (ROM) for 'Depthwise CNN' or 'Pointwise CNN' use. By storing the weight in specific ROMs, the computation units only need to wait for the temporary data from each layer. Namely, the weight parameters will not occupy the DRAM memory bandwidth.

After the calculation, the featured image will be stored in 'Pong RAM' and then written into 'DDR4' through 'DDR Controller'. Since there is no way to store all the feature images of the single-layer model due to the size of 'Ping RAM' and 'Pong RAM', it is necessary to repeatedly access 'DDR4'. When 'Depthwise CNN' completes the calculation, it will notify the 'Controller' to enable the 'Pointwise CNN' to operate. The action of 'Pointwise CNN' is similar to that of 'Depthwise CNN'. To further improve the throughput, we use different PEs for 'Depthwise CNN' and 'Pointwise CNN' to maximize the bandwidth.

After the HGR-Lite model completes, 'Controller' will enable the 'SoftMax' to read the 11 feature results to find the maximum value and then transmit the result to 'Plot Ans' to display the recognition result on the screen.

B. DEPTHWISE CNN

Since the data bandwidth of DDR4 in the ZCU106 board is 512-bit length, in the case of 16-bit feature data, the same address can store up to 32 channels of data. Therefore, in the design of the convolution operation, we use 32 calculation units (PEs) to process 32 channels of data in parallel. Since the convolution kernel size is fixed to 3×3 in the HGR-Lite model, the design of the PE is composed of 9 multipliers and an adder tree to calculate the convolution result.

The 'Depthwise CNN' architecture is shown in Figure 5. 'Flow Controller' receives the control signal (CS) from the

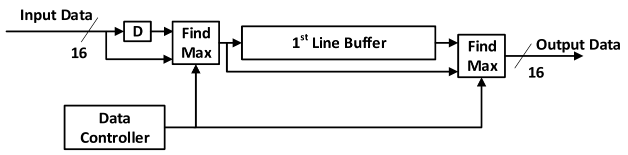

FIGURE 7. Architecture of max-pooling.

TABLE 2. Comparisons of related models.

Model	Accuracy	Input Size	Model size
ResNet-50[39]	0.8138	224x224	99MB
MobileNet[31]	0.8650	224x224	16MB
HGR-Net[26]	0.8810	320x320	2.4MB
Our work	0.8925	200x200	1.07MB

TABLE 3. Quantitative performance of HGR-lite.

FP	32bits	16bits	8bits
Top-1	0.8925	0.892	0.88

'Controller', and CS included such as the start signal, the size of the input feature image, the number of channels, and the number of convolution kernels to control the reading and calculation of 'Depthwise CNN'. In 'Depthwise CNN', the initial address of the 'DDR4' is 640000. It is to avoid overwriting the feature image with 'Pointwise CNN'.

'Depthwise CNN' reads the input data and weights first. The input data is composed of 32 channels to form a 512-bits data format. It needs 9 clock cycles to read and store into the 32-channel 'Input Buffer'. At the same time, the weights are composed of a 3×3 convolution kernel to form a 144-bit data format. It requires 32 clock cycles to read and store to the 32-channel 'Weight Buffer'. When the data is ready, 32 PE operations are executed, and the results of 32 channels are stacked into 512 bits and output.

In PE calculation, 'Input Buffer' will shift the register data of D0, D1, D3, D4, D6, and D7 into D1, D2, D4, D5, D7, D8, and will read the next 3 data into the D0, D3, and D6 registers for pipeline operation. At this time, only three clock cycles are needed to read. However, due to the limited size of 'Ping RAM' and 'Pong RAM', it is impossible to store all the temporary feature data. Therefore, 'Ping RAM' will only read the input data of the maximum storage limit of 'Pong RAM'. When 'Pong RAM' is full, the data will be stored in 'DDR4', and the next input data in 'DDR4' will be written to 'Ping RAM' to continue the depthwise convolution operation. Finally, 'Depthwise CNN' will send a completion signal to notify 'Controller' to perform 'Pointwise CNN'.

C. POINTWISE CNN

Similar to 'Depthwise CNN', 'Pointwise CNN' is also designed with 32 PEs to maximize the data bandwidth of DDR4. 'Pointwise CNN' uses a 1×1 convolution kernel to convolve and sum all the channels of each pixel of the input data. In the case an input data containing 32 channels, the PE is designed to use 32 multiplications and the adder tree to perform the convolution operation.

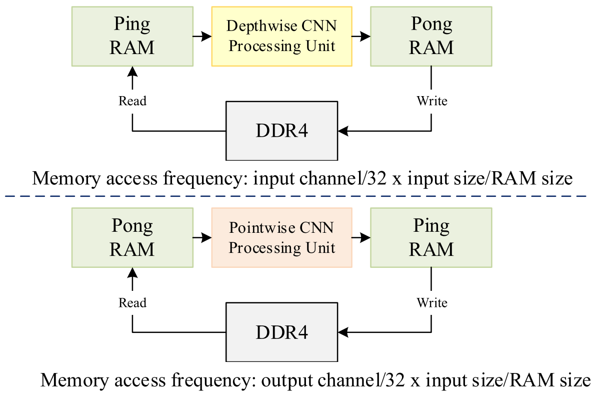
Ping RAM
Depthwise CNN Processing Unit
Pong RAM
Read
Write
DDR4
Memory access frequency: input channel/32 x input size/RAM size
Pong RAM
Pointwise CNN Processing Unit
Ping RAM
Read
Write
DDR4
Memory access frequency: output channel/32 x input size/RAM size

FIGURE 8. Diagram of memory access.

The 'Pointwise CNN' architecture is shown in Figure 6. 'Flow Controller' receives the CS signal from the 'Controller'. It includes the start signal, the input feature map size, the channel, the number of convolution kernels, and the maximum pooling enable signal. 'Pointwise CNN' reads the initial address of the 'DDR4' as 0. 'Pointwise CNN' reads the input data and the weight parameter first. The input data is composed of 32 channels to form a 512-bit data format. 'Pointwise CNN' does not need 'Input Buffer' because it only needs to read one pixel to start the pointwise convolution operation. The weight parameter is composed of a 1×1 convolution kernel of 32 input channels to form a 512-bit data format and 32-bit BN parameters including 16-bit beta and 16-bit gamma. It means that 'Pointwise CNN' needs 32 clock cycles to read and store the parameter to 'Weight Buffer' and 'BN Weight Buffer' for 32 output channels.

When the data is ready, 32 PE operations will be started. Since the input image channel exceeds 32, multiple PE calculations are needed to add up the feature result of an output image channel. For example, when the input image channel is 64 after the first 32 PEs are calculated, the temporary data will be stored in the Ping RAM first. After the second 32 PEs are completed, the Ping RAM data will be read for add-up.

The architecture of the 'Max-Pooling' is shown in Figure 7. We use two comparators and a line buffer to achieve max-pooling operations. It can be directly combined with the convolution hardware accelerator. We designed it with time efficiency without reading data from memory again. This design has better hardware utilization and lower access time.

As the BN operation is performed, we use the beta and gamma parameters to perform the multiply-accumulate operation, and finally, store the 32-channel results of the output image into 'Ping RAM'. And then we transfer it to 'DDR4' through the 'DDR Controller'. All these modules include the PE, the max-pooling, and BN operations performed in the pipeline.

Finally, for softmax operation, we simply choose the largest value of the 11 gestures with comparators for our

TABLE 4. Memory access analysis.

Layer	Input Shape	Output Filter	DCNN Repeat time	PCNN Repeat time	Data access time (300MHz)
SC_bn32_0	200x200x3	32	8	8	0.528ms
SC_bn32_1	200x200x32	32	8	8	0.528ms
SC_bn32_2	200x200x32	64	8	16	0.462ms
SC_bn64_0	100x100x64	64	4	4	0.264ms
SC_bn64_1	100x100x64	64	4	4	0.264ms
SC_bn64_2	100x100x64	128	4	8	0.231ms
SC_bn128_0	50x50x128	128	4	4	0.132ms
SC_bn128_1	50x50x128	128	4	4	0.132ms
SC_bn128_2	50x50x128	256	4	8	0.116ms
SC_bn256_0	25x25x256	256	8	8	0.066ms
SC_bn256_1	25x25x256	256	8	8	0.066ms
SC_bn256_2	25x25x256	128	8	4	0.05ms
SC_bn64_0	13x13x128	64	4	2	0.008ms
SC_bn64_1	13x13x64	64	2	2	0.004ms
SC_bn64_2	13x13x64	32	2	1	0.003ms
SC_bn32_0	6x6x32	32	1	1	--
SC_bn32_1	6x6x32	32	1	1	--
SC_bn32_2	6x6x32	16	1	1	--
SConv16	3x3x16	16	1	1	--
SConv11	1x1x16	11	1	1	--
				Total	2.854ms

prediction instead of realizing the actual softmax architecture. This method saves a lot of hardware resources but does not affect the final prediction.

V. HARDWARE IMPLEMENTATION RESULTS

A. IMPLEMENTATION RESULT AND QUANTITATIVE ANALYSIS

We tested the performance of HGR-Lite using the OUHANDS test dataset [17] and compared it to other works using the same dataset based on the same bit precision (FP32), such as the ResNet-50, MobileNet, and HGR-Net models. The experimental results of ResNet-50 and MobileNet are obtained in [17], and HGR-Net is focused on the HGR task. The comparison results are shown in Table 2. It shows HGR-Lite has a gesture recognition rate of 89%. It runs at the smallest model size with the smallest input size.

To effectively reduce the amount of memory usage, we refer to the experimental results of [20]. In four different network models, they use 16-bit weight parameters, and the error for accuracy is negligible. The performance of our HGR-Lite quantification is shown in Table 3. It is the result of fine-tuning training. It shows that the loss of accuracy is very small when using 16-bit fixed-point numbers so 16-bit fixed-point is used to quantify weights and feature data.

In summary, we use the hand segmentation model as an attention model to train the part of the HGR models first. This strategy allows these convolutional layers to achieve the function of extracting hand features. As a result, it can achieve better recognition for training the entire HGR model. We did not use the entire model since our goal is to reduce the number of parameters and calculations. This result is helpful to implement on embedded systems.

B. MEMORY ACCESS AND RECOGNITION LATENCY

In our proposed HGR-Lite model, the maximum feature data of $200 \times 200 \times 32 \times 16$bits=19.5Mb is generated. The on-chip memory is not enough to store all the feature data. Therefore, 'Ping RAM' and 'Pong RAM' are designed to have a maximum allowable size of $200 \times 25 \times 512$bits =2.44Mb, which is 1/8 of the input image resolution. Therefore, when 'Ping RAM' or 'Pong RAM' is full, the data needs to be stored in 'DDR4' first, and then the required input data is read from 'DDR4' to 'Ping RAM' or 'Pong RAM'.

The data bandwidth of 'DDR4' is 512 bits, and each address can store up to 32 channels of data. Therefore, when the channel of the input feature map is larger than 32, the operation of accessing 'DDR4' needs to perform multiple times as shown in Figure 8. In depthwise convolution, the number of times that DDR4 needs to be accessed is calculated according to the input image size and channel. In pointwise convolution, the number of accesses is calculated according to the input image size, channel, and output image channel, and the detailed access analysis is shown in Table 4. We used DDR4's operating frequency of 300MHz, and a total of 2.854ms is required to access all the feature data, which is acceptable in real-time processing.

C. FPGA DESIGN RESULT AND COMPARISON

Since there is no related work on the neural network hardware accelerator of HGR, we compare our design with the related work of depthwise separable convolution hardware accelerators. The comparison is shown in Table 5. In our design, 32 PEs are used for synthesis. Also, the DSP units and other hardware resources are largely utilized. The synthesis results show that 70~80% of hardware usage has been used. To achieve an efficient hardware architecture design, it is necessary to maximize the utilization of hardware resources and plan the access schedule of the on-chip memory and off-chip memory to achieve the desired performance. In our design, we can achieve 65.6 GOPS and 52.6 FPS performance.

In other related works, [22], [23], [25], [26], [27] also proposed a hardware accelerator with depthwise separable convolution. Reference [22] proposed the RR-MobileNet model to reduce redundant models, which is 25 times smaller than AlexNet's model. The hardware accelerator is implemented on the Xilinx XCZU9EG FPGA with a performance of 127.4 FPS and 91.2 GOPS. However, [22] uses a lot of on-chip memory. Different evaluation boards have great differences in on-chip memory which affects performance significantly. To incorporate the on-chip memory into the evaluation, we use the amount of calculation per Mb (Throughput per Mb, TPM) to quantify the performance of the hardware accelerator. Since Altera's BRAM unit is 20Kb, which is different from Xilinx's BRAM unit of 36Kb, we convert BRAM to Mb as the evaluation standard. Therefore,

TABLE 5. Comparisons of related work.

	This work	2018 [22]	2018 [23]	2020 [25]	2020 [26]	2021 [27]
Platform	Zynq XCZU7EV	Zynq XCZU9EG	Arria 10 SOC	VC707	Zynq 7045	Arria 10 SOC
Network	HGR-Lite	RR-MobileNet	MobileNetV2	Shufflenet	S-Mobile Net	MobileNetV2
Operation Frequency	100MHz	150MHz	133MHz	200MHz	125MHz	200MHz
Arithmetic Precision	16bits	8bits	16bits	18bits	16bits	8bits
BRAM	266.5 (85.42%)	864.5	1,844	1027	64	769
BRAM(Mb)	9.37	30.4	36	32.1	2.25	15.02
DSP	1,515 (87.67%)	1,452	1,278	1926	385	607
FF	153,736 (33.36%)	55K	NA	117091	NA	NA
LUT	166,040 (72.07%)	139K	NA	198120	69666	NA
LUTRAM	47,196 (46.38%)	NA	NA	NA	NA	NA
ALMs	NA	NA	81,753	NA	NA	207k
FPS(Hz)	52.6	127.4	266.6	787.4	240	222.2
Throughput (GOPS)	65.6	91.2	170.6	107.9	33.6	188.17
TPM (GOPS/Mb)	7.01	3	4.74	3.36	14.93	12.52

the TPM of [22] is 3, and our TPM is 7.01. It shows the design [22] in the evaluation of on-chip memory is not as efficient as ours.

Reference [23] implemented a design on the Arria 10 SOC FPGA, using a large amount of on-chip memory to implement large matrix multipliers. Although the FPS and the calculation are better than the other designs, the TPM of [23] is 4.74 and our design is still more efficient in on-chip memory usage. Reference [25] proposed an accelerator for general matrix-matrix multiplication and tested it with Shufflenet. Due to the low computation complexity of Shufflenet, the accelerator achieves high FPS. Because of high memory use, the TPM is just 3.36 which is still 2.086× lower than our work. Reference [26] proposed a high-speed low-cost CNN inference accelerator for depthwise separable CNNs. Although this design can achieve higher FPS, we still have better performance in the throughput comparison. Reference [27] proposed a highly flexible and reconfigurable FPGA hardware accelerator architecture that allows them to obtain higher FPS and throughput, but relatively they also have higher resource usage in BRAM. Based on this comparison, it shows that once our design is using the same size of on-chip memory, the throughput will be better than the other three works. Moreover, in our design, the on-chip memory and PE usage can be re-planned according to different FPGA specifications to achieve optimized hardware performance. Our hardware design can also implement the other kinds of model that uses the depthwise separable convolutional network.

VI. CONCLUSION

This paper proposes the HGR-Lite model with complete software and hardware design. The proposed HGR-Lite uses the hand segmentation model as an attention model to improve its performance of the HGR-Lite model. We use the part of the segmentation model and recognition model to achieve HGR, resulting in a great reduction in the parameters and calculations. Also, the synthesis dataset and iterative training strategy are added to make the model better. In the software part simulation, in the OUHANDS test dataset, an accuracy rate of 89.25% can be achieved.

In the hardware architecture design, we design the pure-hardware end-to-end gesture recognition system on ZCU106 FPGA without any support from the CPU. We use 32 PEs to process 32 channels of data in parallel to maximize the use of DSP units. With storing weight parameters in ROM and the use of the Ping-Pong buffer, the memory has significant improvement. The whole system is implemented on the Xilinx ZCU106 evaluation board. It can achieve 52.6FPS and 65.6 GOPS performance. The performance after quantifying on-chip memory in our design can achieve 7.01 GOPS performance per Mb of on-chip memory and shows a better result than the existing depthwise separable convolutional hardware accelerators.

REFERENCES

[1] P. Kumar, S. S. Rautaray, and A. Agrawal, "Hand data glove: A new generation real-time mouse for human-computer interaction," in *Proc. 1st Int. Conf. Recent Adv. Inf. Technol. (RAIT)*, Mar. 2012, pp. 750–755.

[2] J. Sun, T. Ji, S. Zhang, J. Yang, and G. Ji, "Research on the hand gesture recognition based on deep learning," in *Proc. 12th Int. Symp. Antennas, Propag. EM Theory (ISAPE)*, Dec. 2018, pp. 1–4, doi: 10.1109/ISAPE.2018.8634348.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[4] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: A survey," *Artif. Intell. Rev.*, vol. 43, no. 1, pp. 1–54, Jan. 2015.

[5] C. Mummadi, F. Leo, K. Verma, S. Kasireddy, P. Scholl, J. Kempfle, and K. Laerhoven, "Real-time and embedded detection of hand gestures with an IMU-based glove," *Informatics*, vol. 5, no. 2, p. 28, Jun. 2018.

[6] M. Oudah, A. Al-Naji, and J. Chahl, "Hand gesture recognition based on computer vision: A review of techniques," *J. Imag.*, vol. 6, no. 8, p. 73, 2020, doi: 10.3390/jimaging6080073.

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

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[9] A. Dadashzadeh, A. T. Targhi, M. Tahmasbi, and M. Mirmehdi, "HGR-Net: A fusion network for hand gesture segmentation and recognition," 2018, *arXiv:1806.05653*.

[10] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2016, *arXiv:1610.02357*.

[11] P. K. Pisharady, P. Vadakkepat, and A. P. Loh, "Attention based detection and recognition of hand postures against complex backgrounds," *Int. J. Comput. Vis.*, vol. 101, no. 3, pp. 403–419, Feb. 2013.

[12] G. Plouffe and A. Cretu, "Static and dynamic hand gesture recognition in depth data using dynamic time warping," *IEEE Trans. Instrum. Meas.*, vol. 65, no. 2, pp. 305–316, Feb. 2016.

[13] G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with leap motion and Kinect devices," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 1565–1569.

[14] P. Barros, S. Magg, C. Weber, and S. Wermter, "A multichannel convolutional neural network for hand posture recognition," in *Proc. Int. Conf. Artif. Neural Netw.*, 2014, pp. 403–410.

[15] M. Al-Hammadi, G. Muhammad, W. Abdul, M. Alsulaiman, M. A. Bencherif, and M. A. Mekhtiche, "Hand gesture recognition for sign language using 3DCNN," *IEEE Access*, vol. 8, pp. 79491–79509, 2020.

[16] W. Zhang, J. Wang, and F. Lan, "Dynamic hand gesture recognition based on short-term sampling neural networks," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 1, pp. 110–120, Jan. 2021.

[17] M. Matilainen, P. Sangi, J. Holappa, and O. Silven, "OUHANDS database for hand detection and pose recognition," in *Proc. 6th Int. Conf. Image Process. Theory, Tools Appl. (IPTA)*, Dec. 2016, pp. 1–5.

[18] Y. Ma, N. Suda, Y. Cao, J.-S. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of convolutional neural networks onto FPGA," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–8.

[19] 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*, Feb. 2015, pp. 161–170.

[20] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[22] J. Su, "Redundancy-reduced MobileNet acceleration on reconfigurable logic for ImageNet classification," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2018, pp. 16–28.

[23] L. Bai, Y. Zhao, and X. Huang, "A CNN accelerator on FPGA using depthwise separable convolution," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 10, pp. 1415–1419, Oct. 2018.

[24] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, pp. 303–308, Sep. 2009.

[25] A. Ahmad and M. A. Pasha, "Optimizing hardware accelerated general matrix-matrix multiplication for CNNs on FPGAs," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 11, pp. 2692–2696, Nov. 2020, doi: 10.1109/TCSII.2020.2965154.

[26] Y. Lin, R. Li, W. He, X. Zho, J. He, P. Li, Y. Jiang, L. Liu, N. Wu, and C. Shi, "A high-speed low-cost CNN inference accelerator for depthwise separable convolution," in *Proc. IEEE Int. Conf. Integr. Circuits, Technol. Appl. (ICTA)*, Nov. 2020, pp. 63–64.

[27] X. Wu, Y. Ma, M. Wang, and Z. Wang, "A flexible and efficient FPGA accelerator for various large-scale and lightweight CNNs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 3, pp. 1185–1198, Mar. 2022.

TSUNG-HAN TSAI (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1990, 1994, and 1998, respectively. From 1999 to 2000, he was an Associate Professor of electronic engineering with Fu Jen Catholic University. In 2000, he joined National Central University. Since 2008, he has been a Full Professor with the Department of Electrical Engineering, National Central University. He is currently the Director of the Intelligent Chip and System Center, National Central University. He is the Principal Investigator of the National Program for Intelligent Electronics. His research interests include VLSI signal processing, video/audio coding algorithms, DSP architecture design, wireless communication, and system-on-chip design. He received the Industrial Cooperation Award from the Ministry of Education, Taiwan, in 2003, the Best Paper Award from the IEEE International Conference on Innovations in Bio-inspired Computing and Applications (IBICA), in 2011, and the IEEE International Conference on Innovation, Communication and Engineering (ICICE), in 2015. He has been awarded more than 40 patents and 280 refereed papers published in international journals and conferences. His research team has won many international IC-related student design contest awards, including the ISOCC, in 2015, the TI DSP Asia Design Contest, in 2008, and the ISSCC, in 2011. He has served as the Guest Editor of Special Issues for *Journal of VLSI Signal Processing Systems*. He was the General Co-Chair of IEEE International Conference on Internet of Things, in 2014, and the General Chair of IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), in 2020. He serves as a technical program committee member or the session chair for several international conferences.

YUAN-CHEN HO received the bachelor's degree from the Chung Cheng Institute of Technology, National Defense University, Taiwan, in 2012. He is currently pursuing the master's degree with National Central University, Taiwan. His research interests include deep learning and hardware architecture on hand gesture systems.

PO-TING CHI received the bachelor's degree from National Chiao Tung University, Taiwan, in 2015. He is currently pursuing the master's degree with National Central University, Taiwan. His research interests include computer vision, deep learning, face detection, and image segmentation.