IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 6, 2020, accepted October 7, 2020, date of publication October 14, 2020, date of current version October 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3031055

CNN Acceleration With Hardware-Efficient Dataflow for Super-Resolution

SUMIN LEE, (Graduate Student Member, IEEE), SUNGHWAN JOO, (Graduate Student Member, IEEE), HONG KEUN AHN, (Graduate Student Member, IEEE), AND SEONG-OOK JUNG, (Senior Member, IEEE)
School of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea

Corresponding author: Seong-Ook Jung (sjung@yonsei.ac.kr)

This work was supported by the Samsung Research Funding and Incubation Center for Future Technology under Grant SRFC-IT1802-06.

ABSTRACT The convolutional neural network (CNN)-based super-resolution (SR) has shown outstanding performance in the field of computer vision. The implementation of inference hardware for CNN-based SR has suffered from the intensive computation with severely unbalanced computation load among layers. Various light-weighted SR networks have been researched with little performance degradation. However, the hardware-efficient dataflow is also required to efficiently accelerate inference hardware within limited resources. In this article, we propose the hardware-efficient dataflow of CNN-based SR that reduces computation load by increasing data reuse and increases process element (PE) utilization by balancing the computation load among layers for high throughput. In the proposed dataflow, row-wise pixels in the receptive field are computed by circularly shifting memory addresses to maximize data reuse. The partial convolution is exploited in a layer-based pipeline architecture to relieve intensive computation in a single pipeline stage. The delay-balancing with adjusting parallelism is employed for balancing computations precisely in the overall layers. Furthermore, the inference hardware of CNN-based SR is implemented for 4K ultrahigh definition at 60 fps on a field-programmable gate array (FPGA). For hardware-friendly computation, the quantization of activation and weight is adopted. The proposed hardware shows an average peak signal-to-noise ratio of 36.42 dB in the Set-5 dataset with a memory usage of 53 KB and an average PE utilization of 76.7% in the overall layers. Thus, it achieves the lowest memory usage and highest PE utilization compared with other inference hardware for CNN-based SR.

INDEX TERMS Convolutional neural network, dataflow, field-programmable gate array, inference hardware, 4K ultrahigh definition, process element utilization, super-resolution.

I. INTRODUCTION

Ultrahigh definition (UHD) video resolution has recently been supported by UHD television (TV), internet protocol TV (IPTV), and high-end smartphones. Although UHD resolution (3840 × 2160) provides more immersive visualization than full high-definition (FHD), the use of UHD contents is still limited because legacy contents have already been produced at low resolution (LR), and most video contents are still produced at FHD resolution (1920 × 1080). Therefore, a real-time-based up-scaling technique from FHD resolution or lower resolutions to UHD resolution is required in edge devices [1], [2].

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

The conventional up-scaling technique is bicubic interpolation, which generates high-resolution (HR) images using adjacent pixel data. In this technique, the empty pixel is filled by the estimated values using the weights of the distance between adjacent pixels. The conventional technique is simple and can be easily implemented using hardware. However, its performance is low because it is vulnerable to the noise occurring in the up-scaling process. Hence, artificial intelligence (AI) has been adopted in the up-scaling technique to resolve this problem.

Several deep learning architectures based on AI algorithms have recently been developed in the field of computer vision. Especially, the convolutional neural network (CNN) has demonstrated outstanding performance in computer vision processing areas, such as super resolution (SR),

image classification, and object detection. In particular, CNN-based SR has shown better performance than bicubic interpolation for up-scaling from LR to HR images because it is pre-trained through machine learning with several training sets for performance improvement [3]–[8]. However, its real-time implementation with hardware has limitations because excessive computation and memory capacity are required [28], [29]. Consequently, several lightweight versions of CNN-based SR such as fast super-resolution CNN (FSRCNN) and FSRCNN-s (a small model size version of FSRCNN) have been proposed to reduce data process complexity [3], [7], [8]. These SR networks have shorter layers and fewer weight parameters, which are suitable for hardware implementation. However, it is still difficult to implement FSRCNN and FSRCNN-s with hardware because a large number of frame buffers that store feature map data and excessive computations with complex dataflow are required. In addition, the computation load is severely unbalanced among the layers, which degrades the process element (PE) utilization in the pipeline architecture [29], [32].

Thus, inference hardware for CNN-based SR requires highly efficient dataflow as well as the lightweight SR network to mitigate these restrictions on hardware implementation. Highly efficient dataflow increases data reuse and PE utilization, leading to the achievement of the target system throughput with fewer hardware resources [9]–[15], [30]. Therefore, the research on dataflow optimization is demanded for hardware-efficient acceleration.

This article aims to implement the inference hardware of CNN-based SR with hardware-efficient dataflow. The dataflow is optimized in the layer-based pipeline architecture. A circularly shifted dataflow in the row-wise receptive field is proposed for reducing the required memory capacity. In addition, a partial convolution in the compute-intensive layer and delay balancing in the overall layers using parallel computing factors are proposed for maximizing the PE utilization.

The remainder of this article is organized as follows: The proposed CNN-based SR networks and suitable architectures for hardware implementation are introduced in Section II. The memory-efficient dataflow with high PE utilization is proposed for the layer-based pipeline architecture in Section III. Section IV describes the proposed hardware architecture composed of multiply–accumulate (MAC) cores and dual-port memory. The experimental results for bit precision analysis, SR performance, and hardware architecture are presented in Section V. Finally, this article is concluded in Section VI.

II. PREVIOUS WORKS

Several SR networks have been developed based on the CNN structure. Although these networks have outstanding performance with a deeper network and excessive weight parameters, their implementation using hardware is difficult, as described in Section I. Thus, several studies have focused on reducing computational complexity [3]–[8], [21]–[29]. In this section, we examine various CNN-based SR algorithms and networks in terms of hardware implementation.

Dong *et al*. proposed a deep learning network called super-resolution CNN (SRCNN) that directly learns an end-to-end mapping between LR and HR images for a single image SR [3]. The network is composed of three layers: patch extraction and representation, nonlinear mapping, and reconstruction. Although the network has a lightweight structure, its processing speed for large images is still unsatisfactory. In an extended study, these authors proposed a fast and light version of SRCNN, called FSRCNN [7]. The FSRCNN consists of five layers. Shrinking and expanding layers are added, and the reconstruction layer in SRCNN is changed to a deconvolution layer for direct mapping. Although the FSRCNN demonstrates improved computation speed, the deconvolution layer still has problems such as checkerboard artifacts and a high computational cost [31].

Shi W *et al*. proposed an efficient sub-pixel CNN (ESPCN) [8]. In the ESPCN, the deconvolution layer is changed to an ESPCN layer that handles up-scaling via periodic shuffling at the last layer of the network. As the computational complexity is reduced and each operation is independent, the ESPCN is a more suitable network for hardware implementation.

Alwani *et al*. observed a fusing relationship between adjacent layers in the CNN inference [21]. In a conventional CNN inference, all the intermediate feature maps are stored in an external memory owing to the capacity limitation of the internal memory. Thus, massive external memory access (EMA) occurs, which degrades the power and latency. In a fused-layer network, intermediate feature maps are stored in the internal memory by fusing two or more convolutional layers into a single layer, leading to a reduction in the receptive field to be stored.

Kim Y *et al*. implemented CNN-based SR hardware with 4K UHD resolution at 60 fps on a field-programmable gate array (FPGA) [28]. They proposed a 1D horizontal convolution that transforms a square receptive field into a rectangular receptive field by revising the filters in the network. A rectangular receptive field is more suitable for hardware implementation because it reduces the line memory required to store intermediate feature map data. They adopted a residual connection in their CNN that uses the result of bicubic interpolation to fill up lost information due to a narrow receptive field. In addition, they proposed the compression of intermediate feature maps to reduce the required memory. Although intermediate feature map memory is reduced by compression and 1D horizontal convolution, a large internal memory is still required.

Lee *et al*. implemented a CNN-based SR processor with FHD resolution at 60 fps for mobile devices [29]. They proposed a selective caching-based layer fusion method to minimize EMA. They expanded the output patch size of the final layer for reusing adjacent output pixels in the fused layer. Consequently, although the EMA is slightly increased, the required internal memory decreases because adjacent data

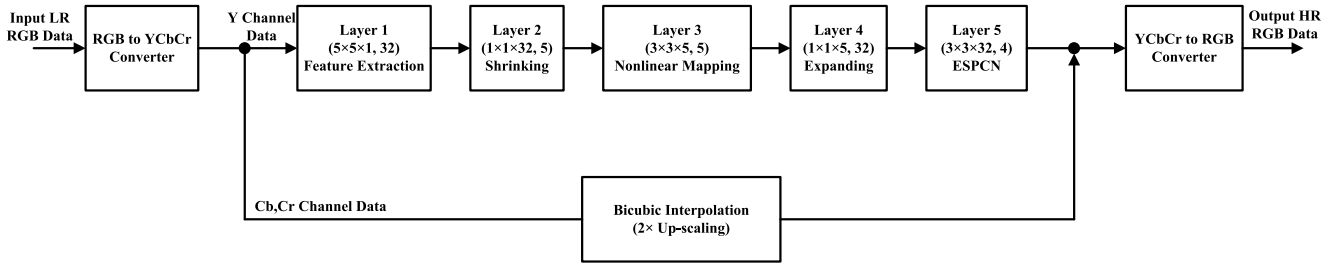

FIGURE 1. Overall block diagrams of the proposed CNN-based SR network from input RGB data to output RGB data. The proposed CNN-based SR network is based on FSRCNN-s, and the deconvolution layer is changed to ESPCN. The numbers in the block diagram at each layer indicate the filter size: They indicate the length of the row, column, and channel and the number of filters, respectively.

are reused for the next operation. In addition, a memory compaction scheme that decreases the window of the receptive field toward the output layer is proposed. The processor has a cyclic ring core architecture to maximize the PE utilization for improving the performance for limited hardware resources. However, the difference in the computation loads in the CNN layers is not considered.

In summary, to implement hardware efficiently, recent studies have focused on not only revising a CNN-based SR network to accelerate inference but also decreasing feature map memory and increasing PE utilization based on the pipeline architecture [16]–[20].

III. PROPOSED MEMORY-EFFICIENT DATAFLOW WITH HIGH PE UTILIZATION FOR CNN-BASED SR

In the inference process of the CNN, the LR image is passed from the input layer to the output layer. All the pixels of the intermediate layer have a data dependency between adjacent layers, which negatively affects parallel processing. Therefore, an efficient dataflow strategy is required for inference hardware with limited resources to perform real-time operations.

A. OVERVIEW

SRCNN and FSRCNN have approximately 8K and 12K parameters, respectively. Thus, they are too large to be implemented with hardware for real-time operations. Hence, a smaller version of FSRCNN, called FSRCNN-s, is used in this study to reduce the number of parameters because it has approximately 4K parameters with an insignificant reduction in the peak signal-to-noise ratio (PSNR) compared with that of the FSRCNN [7]. Although the parameters are decreased from 12K to 4K, it is still challenging to implement FSRCNN-s using hardware for real-time operations. The deconvolution layer is changed to ESPCN to reduce the number of parameters further. Consequently, the number of parameters in the proposed SR network becomes 2.575K, indicating a reduction of approximately 36% compared with that of FSRCNN-s.

In the proposed SR network, input LR RGB data are converted to YCbCr data, which are another expression of color space. The Y channel data are processed by the CNN because

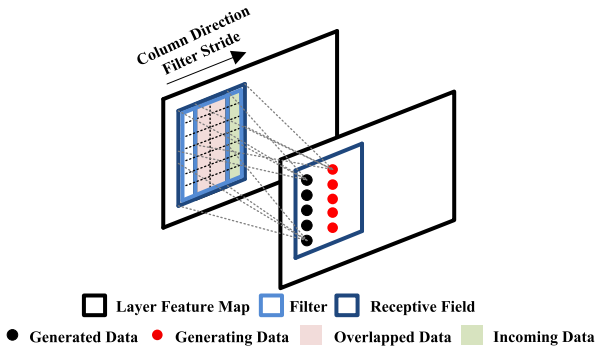

FIGURE 2. Example of inference process in row-wise receptive field with 3 × 3 filter stride.

the proposed SR network is pre-trained for the Y channel, whereas the Cb and Cr channel data are processed by bicubic interpolation with a scaling factor of 2 because they simply need to be converted to RGB. Fig. 1 shows the overall process of the proposed SR network with network information. Filter information is indicated correspondingly in each layer: the lengths of the rows, columns, and channels, and the number of filters. As mentioned previously, the last layer is changed from the deconvolution layer to the ESPCN layer. The Y channel data that pass the ESPCN layer are combined with the Cb and Cr channel data for converting YCbCr to RGB.

The activation occurs in the final operation at each layer, which is the result of convolutional computations such as MAC, rectified linear unit (ReLU), and bias. The fixed-point representation of the activation and weight in the overall SR network is determined by varying the word, integer, and fraction lengths. The detailed results are presented in Section V.

B. CIRCULARLY SHIFTED DATAFLOW IN ROW-WISE RECEPTIVE FIELD

The process of CNN inference involves generating the output pixel data of the next layer through a convolutional operation between the dataset and the filter decided by pre-training. The next data are generated by the filter stride, which is composed of a weight set.

Fig. 2 conceptually illustrates the process of CNN inference between two adjacent layers. The 3×3 filter strides to the column direction in a row-wise receptive field. The column

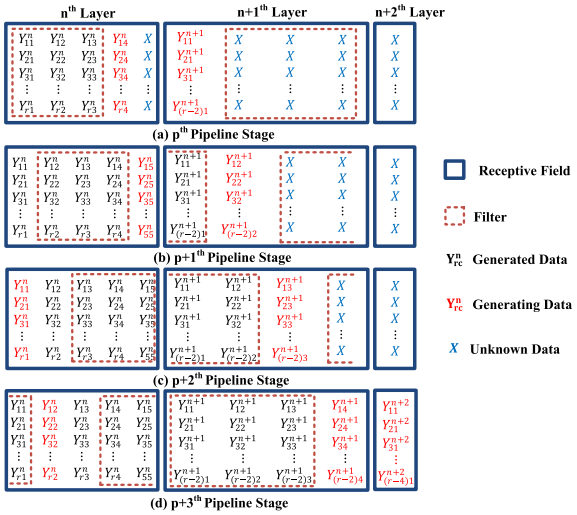

FIGURE 3. Row-wise data reuse method with layer pipeline architecture between three adjacent layers from the p^{th} to $p+3^{th}$ pipeline stage.

data indicated by green (Incoming Data) are newly generated from the previous layer, whereas the others indicated by pink (Overlapped Data) are reused to generate the output pixels in the next layer. The overlapped area is proportional to the row length of the receptive field and the number of strides toward the column direction. The image size of FHD, which is the input LR image, is $1,920 \times 1.080$. The column length is greater than the row length. Thus, the reuse of row-wise data is advantageous because of the large number of column strides.

Fig. 3 shows a detailed illustration of the circularly shifted dataflow with a 3×3 filter between three adjacent layers. At the p^{th} pipeline stage, $r \times 3$ data in the receptive field at the n^{th} layer are calculated to generate the column data of the $n+1^{th}$ layer, where r is the row length in the receptive field at the n^{th} layer. The generated data are indicated in red font. Simultaneously, the 4$^{\text{th}}$ column data at the n^{th} layer are generated from the $n-1^{th}$ layer. As the data process is based on a layer-based pipeline, the unknown data of the $n+1^{th}$ layer are processed with a 3×3 filter. Note that the write and read addresses do not overlap in the same receptive field to prevent data conflicts. At the $p+1^{th}$ stage, all the filters at each layer stride to the column direction to generate the next data. Therefore, the data generated in the previous stage are processed in the current stage. The column data in the $n+2^{th}$ layer are still unknown because the data processed with the filter at the $n+1^{th}$ layer are not yet filled up at the $n+1^{th}$ layer. Consequently, the column data in the $n+2^{th}$ layer are filled after three pipeline stages from the initial stage.

The proposed circularly shifted dataflow in the receptive field has the advantage of reducing the computation load from the matrix to the column vector at each layer. In addition, unnecessary memory capacity is eliminated because only the data in the receptive field need to be stored. The architecture with memory address scheduling is described in detail in Section IV.

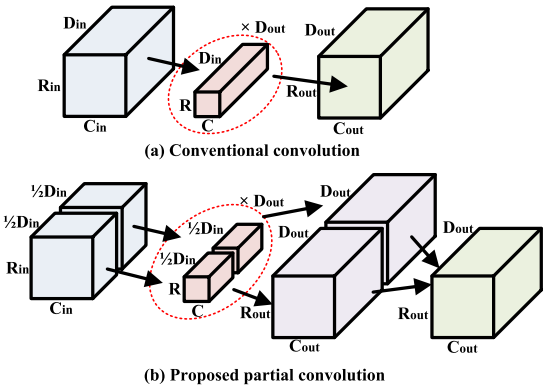

FIGURE 4. Conceptual illustration for the proposed partial convolution: (a) Conventional convolution and (b) proposed partial convolution.

C. PARTIAL CONVOLUTION IN A LAYER-BASED PIPELINE

The convolutional computation is dependent on the filter size. When the filter size is extremely large, the computational time increases linearly. Thus, a method for relieving the filter dependency, called grouped convolution, is required. The grouped convolution was introduced in AlexNet [33], in which the channel dimension and the number of filters are separated into several groups. The separated groups constitute independent subdivided convolutions of a small size, leading to parallelism. The computational cost is defined by the filter size, the number of channels, and the filters at each layer. In the grouped convolution, the entire computational cost is inversely proportional to the number of groups. This is because, as the number of groups increases, the reduced number of channels and filters quadratically decreases the computational cost, whereas the increased number of subdivided convolutions linearly increases the computational cost.

However, the grouped convolution is known to yield a low accuracy when used in regression problems such as SR. Moreover, the accuracy degradation becomes severe when the network has fewer parameters and shorter layers because a small correlation in a lightweight network is also critical to the accuracy. A residual connection is often adopted to maintain the accuracy when grouped convolution is used. Despite the residual connection, additional MAC processing and memory capacity for bicubic interpolation are still required.

Thus, a partial convolution is proposed to reduce the filter dependency in the SR network. The proposed partial convolution is conceptually illustrated in Fig. 4. The input feature map and filter, indicated by blue and red blocks, are separated into two parts. The separated parts individually form independent convolutions, which enable separable pipeline stages. The independent partial products, indicated by purple blocks, are combined at the final stage. In contrast to conventional grouped convolution, the proposed partial convolution does not reduce the number of filters to maintain the full connection between the feature maps and filters because this could degrade the accuracy by losing some connections in the SR network due to regression problems. As the network

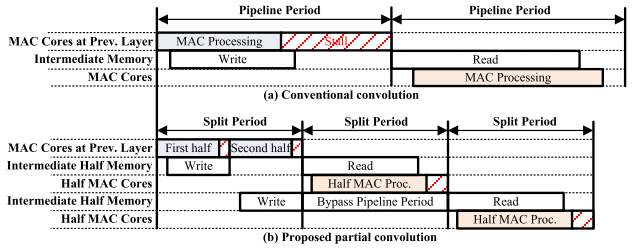

FIGURE 5. Timing diagram for partial convolution in the compute-intensive layer: (a) Conventional convolution and (b) proposed partial convolution.

is not revised in the proposed partial convolution, the computational cost is the same in both the conventional convolution and the proposed grouped convolution. However, the subdivided convolutions are calculated along several pipeline stages in the proposed partial convolution, leading to high PE utilization by relieving the intensive computation load without accuracy degradation. Although the proposed partial convolution requires twice the partial output feature maps compared with the conventional convolution, the increase in memory is acceptable because the doubled partial products are required for only a column vector of the receptive field in the adopted circularly shifted dataflow. As the proposed partial convolution has the flexibility to separate the pipeline stages, the computation load can be balanced among the layers, leading to an efficient dataflow for implementing hardware.

Fig. 5 shows the timing diagram of the partial convolution in a layer-based pipeline architecture. First, the input channels are separated into two groups. The first group is pre-processed at the next pipeline stage. Simultaneously, the second group is bypassed. At the next pipeline stage, the second group is post-processed and combined with the first group. The pipeline stall, indicated by the red diagonal pattern, is significantly decreased in the proposed partial convolution than in the conventional convolution. The proposed partial convolution is a more suitable dataflow because it halves the computation load in a single pipeline stage at the compute-intensive layer without SR performance degradation.

D. DELAY-BALANCING OPTIMIZATION

As mentioned previously, the inference of CNN-based SR has a significantly unbalanced computation load, which degrades the PE utilization in a layer-based pipeline architecture. Thus, the compute-intensive layer is balanced by the proposed partial convolution, which relieves the unbalanced computation load efficiently. However, a more precise balancing method is still required. Therefore, delay-balancing optimization using parallel calculating factors is proposed for maximizing the PE utilization.

The output pixel of the $n+1^{th}$ layer is obtained as follows:

$$Y^{n+1} = \sum_{x=1}^{k_x^n} \sum_{y=1}^{k_y^n} \sum_{z=1}^{k_z^n} Y^n(x,y,z)\, W^n(x,y,z) + B^n \qquad (1)$$

where Y is the feature map data and W is the weight set with data location. k_x, k_y, and k_z are the filter lengths of the rows, columns, and channels, respectively. B is the bias set and the superscript indicates the number of layers. The time required to generate a pixel at the $n+1^{th}$ layer with fully serial MAC is expressed as

$$T^{n+1} = \frac{k_x^n k_y^n k_z^n}{f} k_n^n \qquad (2)$$

where T is the processing time, f is the operating frequency, and k_n is the number of filters. Notably, the receptive field is not included because the data in the receptive field are processed at once owing to the reuse of row-wise data. The processing time is revised as follows because the memory and reset latencies of the accumulator have to be considered.

$$T^{n+1} = \frac{\left(T_{read}^n + k_x^n k_y^n k_z^n + T_{write}^{n+1} + T_{reset}^n\right)}{f} k_n^n \qquad (3)$$

where T_{read} and T_{write} are the read and write latencies of the feature map memory, respectively. T_{reset} is the reset latency of the accumulator in the MAC core. Parallel calculating factors are considered to balance the processing delay at each layer.

$$T^{n+1} = \frac{\left(T_{read}^n + \frac{k_x^n}{p_x^n}\frac{k_y^n}{p_y^n}\frac{k_z^n}{p_z^n} + T_{write}^{n+1} + T_{reset}^n\right)}{f} \frac{k_n^n}{p_n^n} \qquad (4)$$

where p_x, p_y, p_z, and p_n are the parallel calculating factors of the rows, columns, channels, and the number of filters, respectively. Here, p_x is the same as k_x, and p_y is equal to one because the circularly shifted dataflow in the row-wise receptive field is applied.

$$T^{n+1} = \frac{\left(T_{read}^n + k_y^n \frac{k_z^n}{p_z^n} + T_{write}^{n+1} + T_{reset}^n\right)}{f} \frac{k_n^n}{p_n^n} \qquad (5)$$

The pipeline period is determined by the longest processing time among the layers. As the memory throughput should be the same in both the input and output at each layer, the following constraint should be considered.

$$p_n^n = p_z^{n+1} \qquad (6)$$

The parallel calculating factors at each layer are separately tuned to balance the processing delay among the layers, which in turn maximizes the PE utilization. Fig. 6 shows the delay-balancing effect when the parallel calculating factors are properly chosen according to (5) and (6). The normalized processing time among the layers is significantly unbalanced without the delay-balancing optimization, as indicated by the blue line in Fig. 6. The MAC cores at the third layer should be stalled for approximately 1.5 times longer than its normalized processing time, which is approximately 0.4 because the pipeline period is determined by the 1st and 4th layers, which have the longest processing times. When the delay-balancing optimization is adopted for the proposed architecture, as indicated by the red line, the stall time at the third layer is reduced by approximately 5 times, and the overall processing time is balanced among the layers.

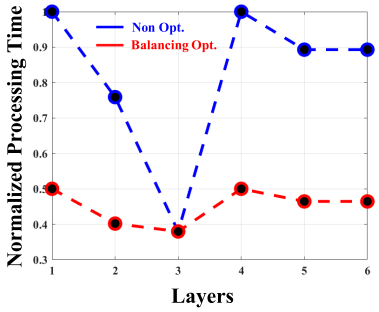

FIGURE 6. Comparison of processing times between nonoptimization and balancing optimization.

IV. PROPOSED ARCHITECTURE

The proposed architecture is composed of MAC cores, single-port random access memory (SPRAM), dual-port random access memory (DPRAM), serial-in parallel-out (SIPO), parallel-in serial-out (PISO), and controllers, as shown in Fig. 7. The intermediate feature maps are stored in the DPRAM because the processing of the feature maps requires simultaneous read and write operations, whereas the weight is stored in the SPRAM. In a pipeline stage, the feature maps in the DPRAM are transferred to the MAC cores, and the MAC operation results are written in the DPRAM of the next layer.

As the main limitation of the conventional inference acceleration technique for CNN-based SR is the shortage of I/O interface to communicate with external memory, the SIPO and PISO are exploited to enable serial communication with external devices. The synchronized SIPO module serially drives the input LR data. Simultaneously, the PISO modules serially transfer the output HR data, which are four times in size compared with the input LR data.

The MAC cores are composed of a multiplier, adders for MAC and bias, a synchronized accumulator, and a ReLU operator. The adders consist of overflow and underflow detectors to prevent over-boundary representation. When multiple multipliers are used in a MAC core, the adders are employed in a tree structure to achieve high speed. The number of multipliers and adders is determined by the row length of the receptive field and the parallel calculating factors as shown in Table 1. The synchronized accumulator is composed of an adder and flip-flops with a synchronized reset. The ReLU operator consists of a most significant bit (MSB) detector

TABLE 1. Detailed information of the proposed Architecture.

Symbol	Layer property	Filter size	RF[a]	PCF[b]
L1	Feature Extraction	(5×5×1,32)	(80,9)	(1,2)
L2	Shrinking	(1×1×32,5)	(76,5)	(2,1)
L3	Nonlinear Mapping	(3×3×5,5)	(76,5)	(1,1)
L4	Expanding	(1×1×5,32)	(74,3)	(1,2)
L5	1st ESPCN	(3×3×16,4)	(74,3)	(2,1)
L6	2nd ESPCN	(3×3×16,4)	(74,3)	(2,1)
L7	Combine ESPCN	-	-	-
Output	-	-	(72,1)	-

[a]RF indicates the lengths of rows and columns in the receptive field, respectively.
[b]PCF indicates the parallel calculating factors of the channels and the number of filters, respectively.

with several MUXs. After bias calculation, the ReLU operator identifies the MSB for clipping the negative values.

Fig. 8 shows the detailed register-transfer level schematic of the L1 MAC core. The thick lines indicate the local bus, and thin lines show activation, weight, and bias signals. At each L1 MAC core, the five multipliers and four adders with tree structure are used because p_x^1 is the same as k_x^1, and p_y^1 is equal to one. The number of multipliers and adders are different at each layer MAC cores. After DPRAM read, the activations indicated by thin blue line are branched from the bus to the individual multipliers in the L1 MAC core. The row-wise activations are convolved during $k_y^1 \times k_z^1 \Big/ p_z^1$ clock cycles. When the activation is transferred to the DPRAM with completion of the accumulation, the read-enable signal (RE) of DPRAM and the reset signal (RST) of the accumulator are simultaneously activated by the clock generated in the local controller, leading to increase in PE utilization.

The address scheduling is used to control the memory access sequence at the DPRAM for implementing the circularly shifted dataflow. Fig. 9 shows an example of memory address scheduling with a 3×3 filter. At the p^{th} pipeline stage, the write address starts from A_0, which is the first address of the DPRAM, indicated by the blue diagonal pattern. Simultaneously, the read address is from A_{c-2} to A_c, indicated by the red diagonal pattern. After a single pipeline period passes, the starting addresses of both the write and read operations are increased by one. Note that the memory address is circularly

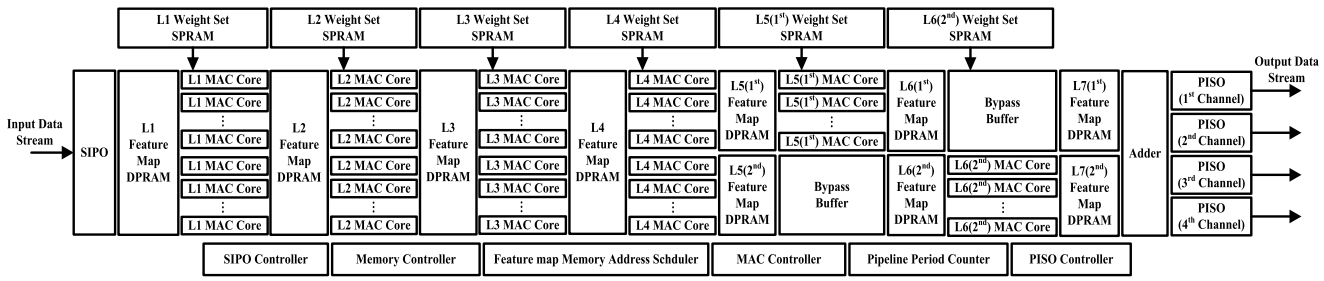

FIGURE 7. Overall CNN-based SR hardware architecture based on a layer pipeline.

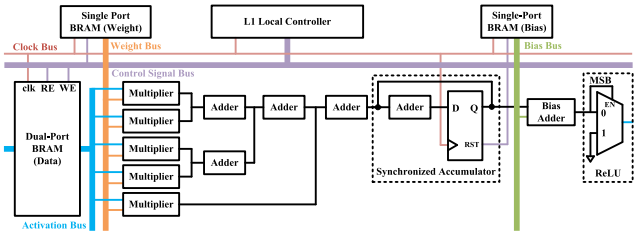

FIGURE 8. Detailed RTL schematic of L1 MAC core.

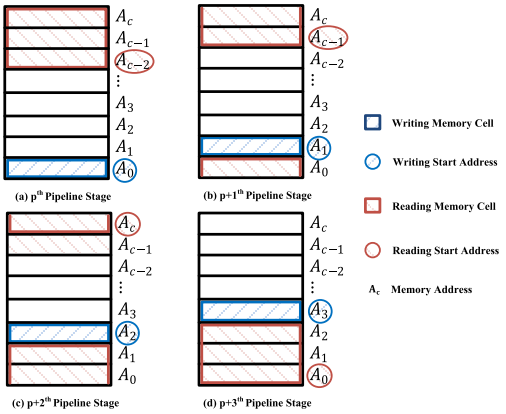

FIGURE 9. Example of address scheduling with a 3 × 3 filter.

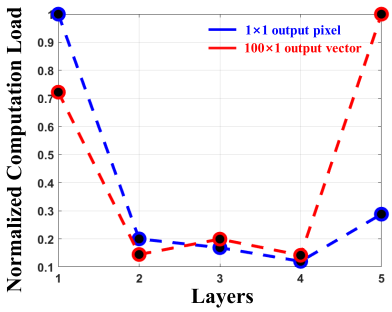

FIGURE 10. Normalized computation load according to the length of receptive field by expanding the output vector of the final layer.

rotated, and the read address is required to be ahead of the write address consecutively. The address order is determined by the priority of the filter direction, and it should be matched with the SPRAM.

The partial convolution is adopted in the ESPCN layer in this study. Fig. 10 shows the normalized computation load at each layer when the 1 × 1 output pixel and 100 × 1 output vector are generated at the final layer. The most compute-intensive layer is the feature extraction layer, indicated by the blue line, when generating an output pixel, whereas it is changed to the ESPCN layer, indicated by the red line, when generating the 100 × 1 output vector. Notably, as the row length of the output vector increases, the ESPCN layer becomes the most compute-intensive layer. As partial convolution is adopted at the ESPCN layer, the DPRAM is divided into two parts from the ESPCN layer, as shown in Fig. 7. The first group at the expanding layer is transferred to the upper DPRAM at the ESPCN layer, whereas the second group is transferred to a lower DPRAM at the ESPCN layer.

Delay-balancing is considered in the proposed architecture. The detailed information on the receptive field and parallel calculating factors in this study is presented in Table 1. The number of MAC cores at each layer is calculated by multiplying the row length of the receptive field with parallel calculating factors. For example, the number of MAC cores at L1 is 152 (76 times 2 cores). From the (5) and Table 1, the processing time of L1 are calculated as 112 clocks. The active processing time is calculated by removing read, write, and reset latencies from (5); The active processing time of L1 is 80 clocks. The average PE utilization in the overall network is calculated by multiplying the active processing time ratio with the PE usage ratio at each layer, in which the active processing time ratio and the PE usage ratio are defined as the ratio of the active processing time to the pipeline period and the ratio of the number of PEs at each layer to the total sum PEs at overall layers, respectively. As a result, the proposed architecture achieves the average PE utilization of 76.7%, allowing higher system throughput with the same hardware resources.

V. EXPERIMENTAL RESULTS

A. QUANTIZATION OF ACTIVATION AND WEIGHT

As the fixed-point number representation has a low complexity in bitwise computation than the floating-point number representation, it is more suitable for implementing the inference hardware of CNN-based SR. The fixed-point representation is defined as [IL, FL], where IL and FL are the integer and fraction lengths, respectively. The word length (WL) of fixed-point representation is calculated as 1+IL+FL when a sign bit is used. Notably, determining IL and FL is important to maintain the accuracy when fixed-point representation is adopted.

Several CNNs are developed on CPU- or GPU-based platforms such as MATLAB, PyTorch, and TensorFlow, which use the 32-bit floating-point representation. Thus, the floating-point representation should be converted to the fixed-point representation for hardware-friendly computation at the inference hardware. Two types of errors occur when floating-point representation is converted to fixed-point representation. The first is the overflow or underflow error related to the IL range of the fixed-point representation. The second is the quantization error related to the FL range of the fixed-point representation. The overflow or underflow error causes more significant accuracy degradation than the quantization error because the number of digits of the IL is larger than that of the FL. In the CNN-based SR architecture, two types of number representations should be determined. The first is activation, which is the representation of intermediate feature maps. The second is weight, which is composed of filters and bias. The activation and weight are converted to fixed-point representation as follows: First, the combination of WL, IL, and FL of the activation is determined to maintain a PSNR comparable to that of the baseline using the least number

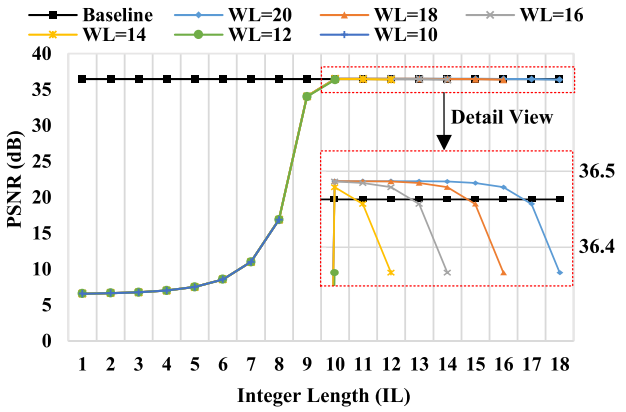

FIGURE 11. PSNR performance plot according to the WL and IL for quantization in Set-5 dataset.

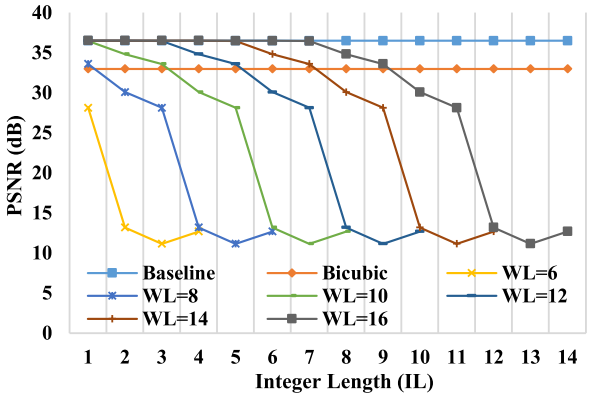

FIGURE 12. PSNR performance plot when the activation is fixed according to the WL and IL of the weight in Set-5 dataset. The experiment is based on WL = 14 and IL = 10 for activation.

of bits without quantizing the weight. Then, the weight is quantized to achieve an acceptable PSNR degradation using the least number of bits.

Fig. 11 shows the PSNR performance according to the WL and IL of the activation for the Set-5 dataset. The baseline is the case without the quantization of the activation and weight. As expected, the case with an inadequate IL shows a large PSNR degradation compared with that of the baseline. An IL of at least 10 is required to maintain the PSNR performance. As the IL increases beyond IL = 10, the PSNR is slightly degraded, as shown in the detailed view in Fig. 11 because the increase in IL with a fixed WL indicates the shrinking of FL, which increases the quantization error. In this study, WL = 14, IL = 10, and FL = 3 are chosen for the activation. This is because this combination not only achieves an acceptable PSNR but also uses a smaller number of bits than the other combinations, which is more suitable for hardware implementation. Fig. 12 shows the PSNR performance according to the WL and IL of the weight when WL = 14, IL = 10, and FL = 3 are used for the activation. A WL of at least 10 is required to maintain a PSNR comparable to that of the baseline. In all the cases, the highest PSNR is achieved when a small number of bits are used for the IL. In this study, WL = 10, IL = 1, and FL = 8 are chosen for the weight because this combination uses the lowest WL while having a comparable PSNR to that of the baseline. In short, the number of bits is quantized from 32-bit to 14-bit for the activation and 32-bit to 10-bit for the weight. The inference hardware can be easily implemented with a reduced-number presentation. Moreover, the PSNR degradation can be insignificant when the WL, IL, and FL are properly chosen. The PSNR degradation is discussed in detail in the next subsection.

B. COMPARISON OF THE SR NETWORKS

Table 2 compares various SR networks, such as the proposed network, bicubic interpolation, SRCNN [3], SRCNN-Ex [4], FSRCNN, and FSRCNN-s [7]. In comparison, the public benchmark datasets are used. The datasets are constructed with Set-5 [34], Set-14 [35], B100 [36], Urban100 [37], and General-100 [7], which are often used for SR benchmark. All experiments are performed with a scaling factor of 2. The PSNR and structural similarity index (SSIM) are used as evaluation metrics.

As described in Section I, CNN-based SR networks show better performances than the conventional bicubic interpolation. The SRCNN [3] has a long processing time for a large image [7], as described in Section II. While the SRCNN-Ex [4] has a better PSNR than the SRCNN [3], it has too many parameters to be implemented with hardware. Although the FSRCNN [7] shows the highest PSNR among these methods, it still has more parameters than the proposed SR network. Even though the FSRCNN-s [7] has a decent PSNR with a reduced number of parameters, the deconvolution layer is not suitable for hardware implementation. The baseline of the proposed SR network has a similar SR performance as the FSRCNN-s, but with an approximately 36% reduction in the number of parameters. As mentioned previously, the activation and weight are quantized for hardware-friendly computation. The average PSNR with quantization is 36.42 dB in the Set-5 dataset, showing a degradation of only 0.07 dB compared with the baseline PSNR of 36.49 dB. As the proposed SR network has a smaller number of parameters with a similar PSNR performance as the FSRCNN-s [7], it is a more suitable SR network for implementation with hardware. Thus, although the proposed SR network has slightly low PSNR and SSIM compared to the other networks, it is suitable for HW implementation thanks to the smaller number of parameters than other networks.

Fig. 13 shows the SR results with a scaling factor of 2 for one of the Set-5 datasets, *butterfly*. In quantitative comparison, the PSNR and SSIM of (c) is 32.21 dB and 0.9621, while the PSNR and SSIM of (b) is 27.47 dB and 0.9146. In qualitative comparison, the proposed hardware shows a comparable image to the original HR image, whereas bicubic interpolation generates a blurred image. In a detailed view, the artifacts are significantly decreased in the proposed SR hardware than in the bicubic interpolation hardware. Fig. 14 shows the other SR results with a scaling factor of 2 for one of the Urban-100 datasets, *img_072*. The PSNR and

TABLE 2. Comparison of various SR methods: Average PSNR and SSIM of three datasets with a scaling factor of 2.

Methods	Bicubic Interpolation		SRCNN [3]		SRCNN-Ex [4]		FSRCNN [7]		FSRCNN-s [7]		Proposed (baseline)		Proposed (HW)	
# of parameters	-		8K		57K		12K		4K		2.58K		2.58K	
Activation bits	-		32-bit		32-bit		32-bit		32-bit		32-bit		14-bit	
Weight bits	-		32-bit		32-bit		32-bit		32-bit		32-bit		10-bit	
Data Set	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Set-5	33.68	0.9304	36.34	0.9521	36.66	0.9542	37.00	0.9557	36.57	0.9531	36.49	0.9538	36.42	0.9529
Set-14	30.48	0.8852	32.18	0.9039	32.42	0.9063	32.63	0.9086	32.28	0.9049	32.29	0.9053	32.27	0.9045
B100	29.58	0.8449	31.11	0.8835	31.36	0.8870	31.50	0.8909	31.23	0.8866	31.18	0.8862	31.18	0.8859
Urban100	26.88	0.8410	29.09	0.8897	29.51	0.8949	29.85	0.9011	29.23	0.8914	29.00	0.8890	28.97	0.8882
General-100	33.25	0.9233	35.93	0.9486	36.30	0.9510	36.68	0.9535	36.12	0.9499	36.05	0.9497	35.96	0.9490

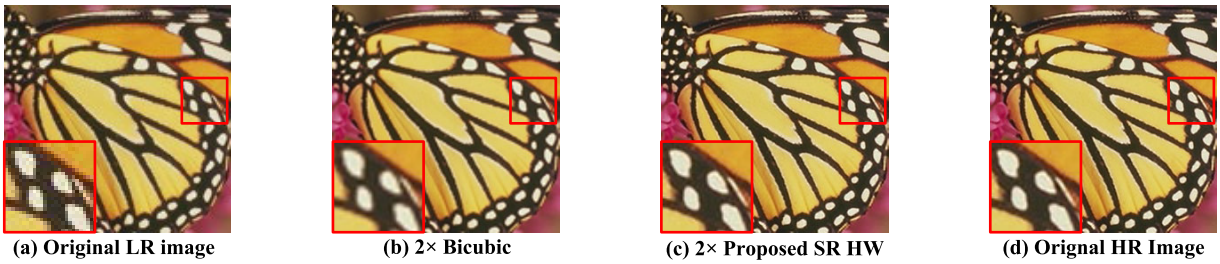
(a) Original LR image (b) 2× Bicubic (c) 2× Proposed SR HW (d) Orignal HR Image

FIGURE 13. SR result with a scaling factor of 2 from one of Set-5 images, *butterfly*.

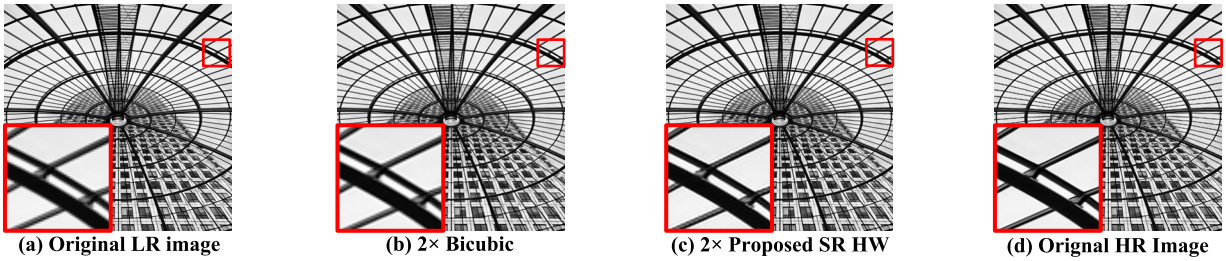
(a) Original LR image (b) 2× Bicubic (c) 2× Proposed SR HW (d) Orignal HR Image

FIGURE 14. SR result with a scaling factor of 2 from one of Urban-100 images, *image_072*.

SSIM of (c) is 22.36 dB and 0.8930, while PSNR and SSIM of (b) is 20.39 dB and 0.8276. The result of proposed hardware is more similar to the original HR image than the bicubic interpolation.

C. COMPARISON OF HARDWARE ARCHITECTURES

Table 3 summarizes the detailed hardware implementations for various SR networks. The hardware architectures of Lee and Park [23] and Kim *et al.* [27] are based on the interpolation method, whereas those of Yang *et al.* [26], Lee *et al.* [29], Kim *et al.* [28], and the current study are based on machine learning. Lee *et al.* [29] implemented inference hardware for mobile devices using the FSRCNN. The hardware supports scaling factors of 2 and 4 for FHD resolution at 25 and 60 fps, respectively. Owing to the cyclic ring core architecture, the hardware achieves a PE utilization of 71.8%. Kim *et al.* [28] implemented two types of hardware on FPGA with the FSRCNN-s: type-1 without the compression of intermediate feature maps, and type-2 with compression. The required memory of type-2 hardware is 194 KB, approximately half that of type-1 hardware. Furthermore, 110K slice look-up tables (LUTs), 102K slice registers, and 151K slice LUTs, 121K slice registers are used in type-1 and type-2 hardware architectures, respectively. Moreover, 1,920 DSP blocks are employed in both types of hardware architectures. As the residual connection is adopted, the average PSNR is higher than that of the others.

Fig. 15 shows the demonstration of the proposed CNN-based SR inference architecture. The proposed architecture is implemented on an FPGA, which is the Xilinx

TABLE 3. Comparison of hardware Architectures for various SR networks.

Publication		Lee [23]	Yang [26]		Kim [27]		Lee [29]	Kim [28][a]	Proposed
SR Methods		Sharp Filters Lagrange	ANR		Edge Orientation Learn Linear Mappings		FSRCNN	FSRCNN-s	FSRCNN-s
Implementation Methods		0.13 um CMOS	Altera EP4SGX530	90 nm CMOS	Xilinx XCKU040	0.13 μm CMOS	65 nm CMOS	Xilinx XCKU040	Xilinx XCVU9P
HW Resources[b]		5.1K	-	1,985K	Slice LUTs : 3,395 Slice Regs : 1,952 DSP Blocks : 108	159K	-	Slice LUTs : 151K Slice Regs : 121K DSP Blocks: 1,920	Slice LUTs : 94K Slice Regs : 19K DSP Blocks: 2,146
Memory Size (KB)		-	235		92		572	194	53
Max. Frequency (MHz)		431	124.4		150	220	200	150	200
Supported Scale		×2, ×3	×2		×2		×2, ×4	×2	×2
PSNR[c] (dB)	Baseline	-	34.00		-		-	36.66	36.49
	HW results	-	33.83		34.78		33.12, 25.64[d]	36.51	36.42
Power (mW/MHz)		-	-		-		1.06	37.91	34.56
PE utilization		-	-		-		71.8%	-	76.7%
Target Resolution		4K UHD (30 fps)	FHD (60 fps)		4K UHD (60 fps)		FHD (25, 60 fps)	4K UHD (60 fps)	4K UHD (60 fps)

[a] Type-2 HW with memory compression
[b] A 2-input NAND gate is counted as one equivalent gate.
[c] The PSNR is average in Set-5.
[d] The PSNR for one of the images in Set-5, *butterfly*

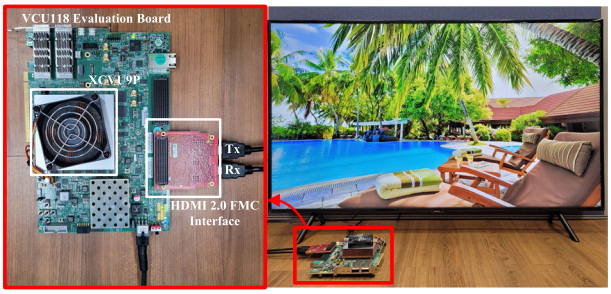

FIGURE 15. Demonstration of the proposed architecture of CNN-based SR with FPGA implementation.

Virtex UltraScale+ VCU118 evaluation board with HDMI 2.0 FMC interface card. The proposed architecture supports a 4K UHD video stream at 60 fps and is implemented at a maximum system operating frequency of 200 MHz. Moreover, 94K slice LUTs and 19K slice registers are used as SIPO, PISO, and adders in MAC cores, and 2,146 DSP blocks are employed as multipliers. Owing to the circularly shifted dataflow, the memory capacity is only 53 KB, including both DPRAM and SPRAM. As partial convolution and delay-balancing optimization are applied, an average PE utilization of 76.7% is achieved.

The proposed hardware achieves the lowest memory usage and highest PE utilization compared with the previous methods. Furthermore, the proposed hardware shows an average PSNR of 36.42 dB in the Set-5 dataset, which is satisfactory considering that the residual connection is not adopted in this study.

VI. CONCLUSION

The CNN-based SR shows more immersive visualization than the conventional bicubic interpolation. However, inefficient dataflow for hardware, excessive computations, and huge memory usage limit its practical use. Therefore, a memory-efficient dataflow with a high PE utilization is proposed with the following characteristics. First, a circularly shifted dataflow, implemented via address scheduling on DPRAM, is proposed for efficient memory usage with high parallelism in a layer-based pipeline architecture. Second, a partial convolution is exploited for relieving the filter dependency at the compute-intensive layer in a layer-based pipeline architecture. The proposed partial convolution does not cause accuracy degradation because it is irrelevant to the network training. Finally, delay-balancing optimization is adopted in the overall layers. Parallel calculating factors are employed to balance the processing time among the layers, considering the constraint of memory throughput.

Furthermore, the inference hardware of the CNN-based SR is implemented on FPGA. The implemented hardware supports up-scaling the video stream from FHD to 4K UHD at 60 fps and shows an average PSNR of 36.42 dB in the Set-5 dataset. The implemented hardware uses the entire memory capacity of 53 KB, which is decreased by approximately 72.7% compared with that of the previous method. In addition, the implemented hardware achieves an average PE utilization of 76.7%, which is approximately 5% higher than that of the previous method. In short, the hardware implemented in this study achieves the lowest memory usage and highest PE utilization compared with the state-of-the-art hardware for CNN-based SR.

As future work, we will extend our study to support real-time CNN-based SR for 8K resolution by fusing circularly shifted dataflow in the row direction combined with multiple clock domains.

REFERENCES

[1] M. Sakurai, Y. Sakuta, M. Watanabe, T. Goto, and S. Hirano, "Super-resolution through non-linear enhancement filters," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2013, pp. 854–858.
[2] J.-S. Choi and M. Kim, "Single image super-resolution using global regression based on multiple local linear mappings," *IEEE Trans. Image Process.*, vol. 26, no. 3, pp. 1300–1314, Mar. 2017.
[3] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Cham, Switzerland: Springer, 2014, pp. 184–199.
[4] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 295–307, Feb. 2016.
[5] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1646–1654.
[6] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 1132–1140.
[7] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Cham, Switzerland: Springer, 2016, pp. 391–407.
[8] W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1874–1883.
[9] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.
[10] M. T. Hailesellasie and S. R. Hasan, "MulNet: A flexible CNN processor with higher resource utilization efficiency for constrained devices," *IEEE Access*, vol. 7, pp. 47509–47524, 2019.
[11] Y. Shen, T. Han, Q. Yang, X. Yang, Y. Wang, F. Li, and H. Wen, "CS-CNN: Enabling robust and efficient convolutional neural networks inference for Internet-of-Things applications," *IEEE Access*, vol. 6, pp. 13439–13448, 2018.
[12] G. Shu, W. Liu, X. Zheng, and J. Li, "IF-CNN: Image-aware inference framework for CNN with the collaboration of mobile devices and cloud," *IEEE Access*, vol. 6, pp. 68621–68633, 2018.
[13] Y. Li and Y. Du, "A novel software-defined convolutional neural networks accelerator," *IEEE Access*, vol. 7, pp. 177922–177931, 2019.
[14] S. Li, Y. Luo, K. Sun, N. Yadav, and K. K. Choi, "A novel FPGA accelerator design for real-time and ultra-low power deep convolutional neural networks compared with titan X GPU," *IEEE Access*, vol. 8, pp. 105455–105471, 2020.
[15] W. G. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24411–24432, 2018.
[16] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 553–564.
[17] S. I. Venieris and C.-S. Bouganis, "FpgaConvNet: A framework for mapping convolutional neural networks on FPGAs," in *Proc. IEEE 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2016, pp. 40–47.
[18] W. Choi, K. Choi, and J. Park, "Low cost convolutional neural network accelerator based on bi-directional filtering and bit-width reduction," *IEEE Access*, vol. 6, pp. 14734–14746, 2018.
[19] C. Park, S. Park, and C. S. Park, "Roofline-Model-Based design space exploration for dataflow techniques of CNN accelerators," *IEEE Access*, vol. 8, pp. 172509–172523, 2020.
[20] X. Hu, Y. Zeng, Z. Li, X. Zheng, S. Cai, and X. Xiong, "A resources-efficient configurable accelerator for deep convolutional neural networks," *IEEE Access*, vol. 7, pp. 72113–72124, 2019.
[21] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.
[22] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.
[23] J. Lee and I.-C. Park, "High-performance low-area video up-scaling architecture for 4-K UHD video," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 4, pp. 437–441, Apr. 2017.
[24] E. Perez-Pellitero, J. Salvador, J. Ruiz-Hidalgo, and B. Rosenhahn, "Accelerating super-resolution for 4K upscaling," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2015, pp. 317–320.
[25] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
[26] M.-C. Yang, K.-L. Liu, and S.-Y. Chien, "A real-time FHD learning-based super-resolution system without a frame buffer," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 12, pp. 1407–1411, Dec. 2017.
[27] Y. Kim, J.-S. Choi, and M. Kim, "2X super-resolution hardware using edge-orientation-based linear mapping for real-time 4K UHD 60 fps video applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 9, pp. 1274–1278, Sep. 2018.
[28] Y. Kim, J.-S. Choi, and M. Kim, "A real-time convolutional neural network for super-resolution on FPGA with applications to 4K UHD 60 fps video services," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 8, pp. 2521–2534, Aug. 2019.
[29] J. Lee, D. Shin, J. Lee, J. Lee, S. Kang, and H.-J. Yoo, "A full HD 60 fps CNN super resolution processor with selective caching based layer fusion for mobile devices," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C302–C303.
[30] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. 37th Annu. Int. Symp. Comput. Archit. (ISCA)*, Saint-Malo, France, 2010, pp. 247–257.
[31] 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," Apr. 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861
[32] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.
[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.
[34] M. Bevilacqua, A. Roumy, C. Guillemot, and M.-L.-A. Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," in *Proc. Brit. Mach. Vis. Conf.* Swansea, U.K.: BMVA Press, 2012, p. 135.
[35] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," in *Proc. Int. Conf. Curves Surf.* Berlin, Germany: Springer, 2012, pp. 711–730.
[36] J.-B. Huang, A. Singh, and N. Ahuja, "Single image super-resolution from transformed self-exemplars," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 5197–5206.
[37] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th IEEE Int. Conf. Comput. Vis. (ICCV)*, vol. 2, Jul. 2001, pp. 416–423.

SUMIN LEE (Graduate Student Member, IEEE) was born in Seoul, South Korea, in 1991. He received the B.S. degree in electronic engineering from Inha University, Incheon, South Korea, in 2016. He is currently pursuing the Ph.D. degree in electrical and electronic engineering with Yonsei University, Seoul.

His current research interests include mixed-mode, low-power circuit, and architecture design for deep neural networks.

SUNGHWAN JOO (Graduate Student Member, IEEE) was born in Seoul, South Korea, in 1989. He received the B.S. degree in electrical and electronic engineering from Korea Aerospace University, Seoul, in 2016. He is currently pursuing the Ph.D. degree in electrical and electronic engineering with Yonsei University, Seoul.

His current research interests include deep learning circuit and architecture design.

HONG KEUN AHN (Graduate Student Member, IEEE) was born in Seoul, South Korea, in 1993. He received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, in 2017, where he is currently pursuing the Ph.D. degree in electrical and electronic engineering.

His current research interests include computing-in-memory circuit design and deep learning architecture.

SEONG-OOK JUNG (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 1987 and 1989, respectively, and the Ph.D. degree in electrical engineering from the University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 2002.

From 1989 to 1998, he was affiliated with Samsung Electronics Company Ltd., Hwaseong, South Korea, where he was involved in specialty memories, such as video, graphic, and window RAM, and merged memory logic. From 2001 to 2003, he was affiliated with T-RAM Inc., Mountain View, CA, USA, where he was the Leader of the Thyristor Based Memory Circuit Design Team. From 2003 to 2006, he was affiliated with Qualcomm Inc., San Diego, CA, USA, where he was involved in high-performance low-power embedded memories, process variation-tolerant circuit design, and low-power circuit techniques. Since 2006, he has been a Professor with Yonsei University. His current research interests include process variation-tolerant, low-power, mixed-mode circuit design, and next-generation memory and technology.