ADAS: A High Computational Utilization Dynamic Reconfigurable Hardware Accelerator for Super Resolution

LIANG CHANG, XIN ZHAO, and JUN ZHOU, University of Electronic Science and Technology of China, China

Super-resolution (SR) based on deep learning has obtained superior performance in image reconstruction. Recently, various algorithm efforts have been committed to improving image reconstruction quality and speed. However, the inference of SR contains huge amounts of computation and data access, leading to low hardware implementation efficiency. For instance, the up-sampling with the deconvolution process requires considerable computation resources. In addition, the sizes of output feature maps of several middle layers are extraordinarily large, which is challenging to optimize, causing serious data access issues. In this work, we present an all-on-chip hardware architecture based on the deconvolution scheme and feature map segmentation strategy, namely ADAS, where all the generated data by the middle layers are buffered on-chip to avoid large data movements between on- and off-chip. In ADAS, we develop a hardware-friendly and efficient deconvolution scheme to accelerate the computation. Also, the dynamic reconfigurable process element (PE) combined with efficient mapping is proposed to enhance PE utilization up to nearly 100% and support multiple scaling factors. Based on our experimental results, ADAS demonstrates real-time image SR and better image reconstruction quality with PSNR (37.15dB) and SSIM (0.9587). Compared to baseline and validated with the FPGA platform, ADAS can support scaling factors of 2, 3, and 4, achieving 2.68×, 5.02×, and 8.28× speedup.

CCS Concepts: • **Computer systems organization → Neural networks;**

Additional Key Words and Phrases: Super resolution, deconvolution, reconfigurable architecture, computation efficiency, data access

ACM Reference format:
Liang Chang, Xin Zhao, and Jun Zhou. 2023. ADAS: A High Computational Utilization Dynamic Reconfigurable Hardware Accelerator for Super Resolution. *ACM Trans. Reconfig. Technol. Syst.* 16, 3, Article 47 (June 2023), 22 pages.
https://doi.org/10.1145/3570927

1 INTRODUCTION

As one of the successful artificial intelligent algorithms, deep convolution neural networks have been widely employed in various tasks, such as object detection, automatic driving, and image classification [4, 14, 20, 44]. Recently, a series of **super-resolution (SR)** works employed deep convolution neural networks, including FSRCNN, IDN, and DRCN [3, 5, 9–12, 17, 18, 21, 43, 45],

This work was supported by the National Natural Science Foundation of China under Grant No. 62104025, and State Key Laboratory of Computer Architecture (ICT, CAS) under Grant No. CARCHB202117.
Authors' address: L. Chang, X. Zhao, and J. Zhou, University of Electronic Science and Technology of China, Chengdu, China, 611731; emails: liangchang@uestc.edu.cn, xinzhao@std.uestc.edu.cn, zhouj@uestc.edu.cn.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
1936-7406/2023/06-ART47 $15.00
https://doi.org/10.1145/3570927

which has improved the reconstruction performance compared to traditional methods [13, 16, 27, 35]. For hardware accelerators, the performance of deep-learning-based SR hardware implementations [15, 22, 25, 26, 33, 36] is also better than that of the traditional method-based SR hardware implementations [2, 23, 27, 39]. However, the deep-neural-network-based SR algorithm contains an up-sampling process like the deconvolution layer, inducing massive computation operations for hardware implementation. Sufficient computational resources are required to accelerate the deconvolution layer to meet the real-time requirement.

Although the deconvolution layer can be optimized on the hardware acceleration with various methods [6, 30, 42], several obstacles are still challenging to be solved. On the one hand, low **processing element (PE)** utilization and data load unbalance can be further explored to improve the computation efficiency. On the other hand, the function of data-intensive SR is to reconstruct images or videos and generate the output with higher resolution. The process of forward inference produces a large amount of output data between different layers due to up-sampling and the large size of the input feature map. In addition, with the development of display technology, ultra-high resolution is required by many types of advanced display equipment, such as 4K and 8K monitors. Consequently, data access becomes a foreseeable bottleneck to obstacle hardware performance. The frequent data access between on-chip logic and off-chip memory may result in extremely high power consumption and low system throughput for bandwidth limitation.

Based on the above observations, several solutions can be employed. For instance, the compression technique can be used to reduce memory overhead [24, 28]. Layer fusion can decrease data access between on- and off-chip memory to relieve the pressure of bandwidth [1]. In addition, various acceleration methods provide solutions to SR, including Winograd convolution and im2col scheme [31, 40, 41] to accelerate computation by reducing the number of multiplications and converting convolution to matrix multiplication. In this article, to further improve the performance of the SR-based hardware accelerator, we explore an end-to-end hardware implementation on FPGA. We develop A high computational utilization Dynamic reconfigurable hardware Accelerator for SR, namely *ADAS*. The *ADAS* is equipped with a hardware-friendly and efficient deconvolution scheme to develop a high-computational-efficiency intelligent SR acceleration system. Also, the all-on-chip strategy is executed to reduce the data movements between on-chip logic and off-chip memory, correspondingly saving the bandwidth. The contributions of this article can be summarized as follows:

- We present an all-on-chip super-resolution hardware architecture, namely *ADAS*, based on the proposed deconvolution scheme and the idea of feature map segmentation. Only the input and output of *ADAS* access with the off-chip, which can avoid the data access between the middle layer and off-chip memory, further improving the system throughput to maintain real-time super-resolution. In addition, the interpolation and color space conversion modules are simplified to further lightweight hardware resources.
- We propose an efficient deconvolution scheme and a dynamic reconfigurable PE, namely *DR_PE*. The *DR_PE* is based on exploring computing resource utilization and load balance. This method can support different scaling factors, achieving nearly 100% PE utilization and eliminating the problem of load unbalance. Correspondingly, a significant end-to-end speedup is achieved in super-resolution. In addition, the proposed output data alignment strategy can realize the sequential output of the feature map after super-resolution and alleviate the data re-arrangement outside the chip. To the best of our knowledge, our deconvolution scheme has achieved 100% computational utilization for the first time without a load unbalance problem.
- We re-train and quantize the SR network to ensure image reconstruction quality while reducing logical resource overhead. In order to adapt to the all-on-chip architecture, several

Table 1. The Structure of FSRCNN

Layer	Operation	Kernel Size	Output Size (MB)[a]
L1	Convolution + PReLu	5×5	442.96
L2	Convolution + PReLu	1×1	94.92
L3	Convolution + PReLu	3×3	94.92
L4	Convolution + PReLu	3×3	94.92
L5	Convolution + PReLu	3×3	94.92
L6	Convolution + PReLu	3×3	94.92
L7	Convolution + PReLu	1×1	442.96
L8	Deconvolution	9×9	7.92

[a]Output feature map size is obtained with the input size of 1080p (1920×1080) in fp32 precision.

optimization strategies are used to re-train the network to obtain a higher **peak signal-to-noise ratio (PSNR)** and **structural similarity (SSIM)**. In addition, quantization is applied to facilitate hardware deployment.

The remainder of this article is organized as follows. Section 2 introduces the basic knowledge of deconvolution and super-resolution and then analyzes the existing problems. Section 3 discusses the designed all-on-chip super-resolution architecture. Section 4 explores the optimization space of deconvolution and proposes the optimization method. Section 5 illustrates the experimental results and detailed analyses. Finally, we conclude this work in Section 6.

2 PRELIMINARY AND OBSERVATION

2.1 Super-resolution

Super-resolution refers to the reconstruction of low-resolution images or videos to obtain high-resolution ones without losing image quality as much as possible [34]. It can be used in many fields such as industrial detection, live broadcast, medicine, and security [19, 37]. Typically, the super-resolution methods contain interpolation, reconstruction, and learning schemes. The learning-based scheme has demonstrated superior results and performance compared to the traditional scheme. Here, the commonly used evaluation metrics for image reconstruction quality of super-resolution include the PSNR and SSIM index [38]. Usually, the PSNR is used to evaluate the SR-based hardware model, where this work employs both the PSNR and SSIM to facilitate comparison.

2.1.1 Observation (I). Super-resolution is different from conventional neural networks like image classification. Traditionally, in the process of forward inference, the feature map size of the next layer decreases, and the amount of data decreases gradually. However, in super-resolution, the amount of data of the output feature map in each layer does not decrease and occasionally increases. In addition, the input feature map processed by SR is generally much larger than that of other CNNs. For instance, the FSRCNN network takes 1080p images as input for the SR system, getting many inter-layer output data. First, the inter-layer data induces a serious data interaction problem to deploy the SR on FPGA, as shown in Table 1. Moreover, such output data of each layer causes high-memory-access power and reduces the system's overall performance. Furthermore, storing all intermediate data on-chip requires a significant memory resource, and it is challenging to find a current FPGA solution to meet the requirement.

2.2 Deconvolution

In a convolution neural network, there are two types of sampling units: down-sampling and up-sampling. For example, pooling is the down-sampling unit. On the contrary, deconvolution belongs

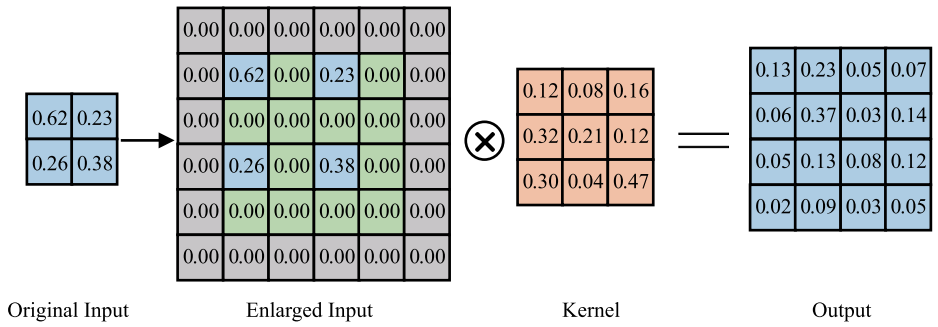

Fig. 1. An example of deconvolution operation. For the enlarged input feature map, the blue, green, and gray boxes indicate the valid data from the original input feature map, the interpolated "0" value, and the padding data, respectively.

to the up-sampling unit, which is used to enlarge the input feature map. With different scaling factors, output feature maps with different sizes can be obtained. The deconvolution unit is common in neural networks such as super-resolution [12] and **Generative Adversarial Networks (GANs)** [46]. The deconvolution layer on hardware is typically developed via interpolation to enlarge the input feature map. The enlarged input feature map is then processed by the traditional convolution method to obtain the final output feature map. Consequently, deconvolution computation is much larger than convolution computation due to input and kernel size enlargement. Figure 1 indicates the computation flow of deconvolution, where the width and height of the input feature map are 2, and the scaling factor is set to 2 (i.e., double the size of the input feature map). Initially, the input feature map is filled with "0" to get an enlarged input feature map with the size of 6×6. Then, a 3×3 convolution is carried out to obtain the output feature map with the size of 4×4, twice larger than the original input feature map.

2.2.1 Observation (II). The above implementation is inefficient since a large number of "0" values should be inserted into the input feature map. The inserted "0" is recognized as invalid computation to the final results. As shown in Figure 1, the proportion of invalid computation is as high as 52.1%. In this article, we discuss the optimization of inserted "0" rather than the optimization of "0" value brought by the padding operation. In addition, the larger scaling factor will induce more invalid computations. Here, the critical issue is eliminating the invalid computation of the deconvolution layer to improve the computation efficiency.

2.3 Structure of FSRCNN

FSRCNN is a high-performance and effective SR network with the learning-based scheme, which is adopted in most SR-based hardware works, as shown in Table 1, where a 5×5 convolution layer is used for feature extraction [12]. Two 1×1 convolution layers are applied to the compression and decompression of the computation, where the former is employed to ensure the computation reduction of four middle 3×3 convolution layers. At the same time, the latter is used to provide sufficient sampling data for deconvolution. In the whole network, the deconvolution layer with 9×9 kernel size accounts for 69.6% of the computation, which needs to be optimized. In this article, we implement and optimize the SR to the FPGA based on FSRCNN.

2.4 Related Work

Based on the above observations, two innovation designs can be considered: improving the computation efficiency of the deconvolution layer and optimizing memory accesses. To improve

the computation efficiency of the deconvolution layer, [7] employed the **transforming deconvolution-to-convolution (TDC)** method. The original deconvolution operation could be converted into multiple convolution operations and reduce the convolution kernel size by compressing the invalid zero computation. For example, the deconvolution with kernel size 5×5 can be decomposed and compressed to four convolution operations with kernel size 3×3, 3×2, 2×3, and 2×2. Correspondingly, the load time is 9, 6, 6, and 4, with the computational utilization of 100%, 67%, 67%, and 44%, respectively, where the loading time is imbalanced and computational utilization is low for hardware implementation. [6] is an extension work of [7], which improved the load balance. Specifically, [6] distributes the one-third computation of convolution 3×3 to convolution 2×2, which made the load time to 6 (9 − 3), 6, 6, and 7 (4 + 3), with the computational utilization of 67%, 67%, 67%, and 78%, respectively. As a result, the loading time balance is improved while the computational utilization is still low.

The computational utilization and load balance can be further improved with both the algorithm and the hardware innovation. [31] proposed a Winograd fast algorithm-based deconvolution layer, reducing the number of multiplications. In addition, the input frames can be divided into multiple blocks, reducing the data capacity of the middle layer feature map, which dramatically reduces the required on-chip memory resources [15, 31]. For bandwidth optimization, [8] developed a bandwidth-efficient architecture based on the layer fusion mechanism, reducing the bandwidth by 98.4%. [28] and [26] proposed a tile-based selective SR chip SRNPU, which can reduce communication bandwidth with external memory by 78.8% by enabling efficient caching of intermediate feature maps with short reuse distance.

We mainly focus on hardware innovation in our *ADAS* architecture. We improve the computation utilization of the deconvolution layer and achieve load balance on the FPGA. All the intermediate data are handled on-chip to reduce the memory accesses and improve the performance of the SR-based hardware accelerator.

3 THE PROPOSED *ADAS* HARDWARE ARCHITECTURE

This section presents the proposed *ADAS* architecture and each component of *ADAS*. The operations and all-on-chip mechanism are discussed. The detailed design of the core computation engine for the deconvolution layer will be provided in the next section.

3.1 Overall Architecture

Figure 2 provides the all-on-chip *ADAS* hardware architecture, including accelerator cores with **dynamic reconfigurable PE (DR_PE)**, **activation/weight (A/W)** distributor, Share Adder Tree with inter-layer buffer, Nearest/T-buffer units, RGB2YCbCr and YCbCr2RGB units, Output Reshaper, **special process unit (SPU),** and Global Controller. Several necessary design components, such as weight/input buffers, **direct memory access (DMA)** unit, and off-chip memory, are also demonstrated in Figure 2. In the *ADAS* architecture, the inter-layer output feature map data can communicate with the inter-layer buffer according to the all-on-chip mechanism without moving into the off-chip memory (❶). The DR_PE locates in a fixed core to accelerate the deconvolution layer (❷), and the A/W distributor is used to balance the computation in different cores (❸). SPU is used for activation, data truncation, and other operations. The Global Controller unit controls the interactions among units in the hardware architecture.

For the end-to-end processing of *ADAS*, the input feature map and weight data are transferred from off-chip memory to the on-chip buffer through the DMA engine. For weight data, we load the entire weight data of the network into the weight buffer at once. The RGB2YCbCr unit preprocesses the input feature map data and then sends to the PE cores and the traditional interpolation unit for subsequent operation, respectively. Before the PE core performs computation, the

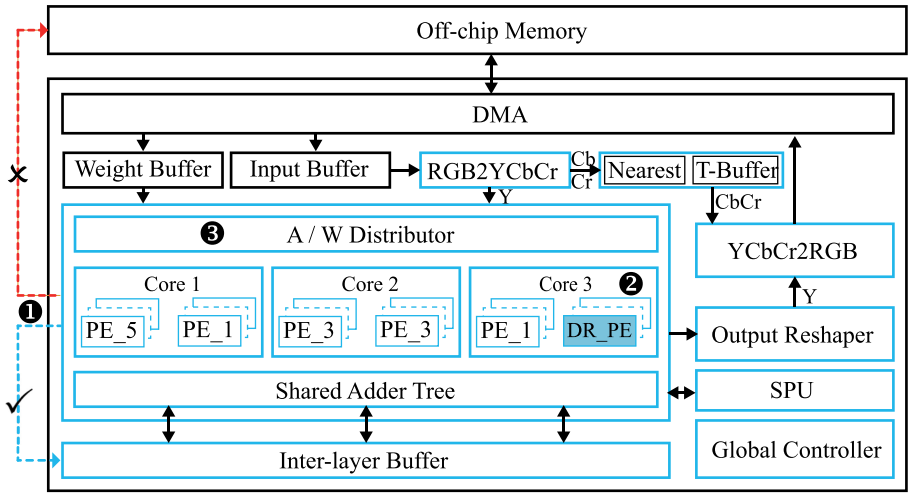

Fig. 2. *ADAS* architecture. ❶ indicates that the all-on-chip mechanism is adopted, and the feature map data output from the middle layer does not interact with the off-chip memory. ❷ indicates that the proposed dynamic reconfigurable PE (DR_PE) is used for the deconvolution operation. ❸ realizes efficient mapping in the process of deconvolution (or convolution).

data to be computed is first distributed to the corresponding PE unit through the A/W Distributer. The PE computation results are added via the Shared Adder Tree and then output to the Output Reshaper. The computation results are processed by the post-processing unit YCbCr2RGB and then output to the off-chip.

3.2 Design Components and Operations

3.2.1 RGB2YCbCr (Preprocessing). For the input RGB image, we first convert it to YCbCr color space. The specific conversion is shown in Equations (1) to (3). We can see that each component coefficient is a fractional value. In order to simplify the computation, we shift all the coefficients to the right by 12 bits, which becomes integer coefficient multiplication:

$$Y = (16 + 0.504G) + (0.257R + 0.098B) \tag{1}$$

$$Cb = (128 + 0.439B) - (0.148R + 0.291G) \tag{2}$$

$$Cr = (128 + 0.439R) - (0.368G + 0.071B). \tag{3}$$

The specific hardware structure is shown in Figure 3. The integer coefficients are stored in the Coefficient Map unit, which remains unchanged during the computation. The input of RGB color space is dispatched to the multiplication unit, multiplied by the corresponding coefficients, and then sent to the adder tree. Before output, the intermediate results are processed by the Data Cut unit, and the lower 12 bits are rounded off. Then, YCbCr results are obtained. Because the human eye is more sensitive to the luminance component Y, we use the neural network method to process the Y component. Thereby, the data of the Y component is dispatched to the PE block. Meanwhile, for the chrominance component Cb and Cr, we use the traditional interpolation method and send them to the nearest unit.

3.2.2 PE Cores (Computation for Y Channel). As shown in Figure 2, only the PE for deconvolution is dynamically reconfigurable. In the whole network, the convolution size includes 1×1, 3×3, and 5×5. Correspondingly, in addition to the DR_PE for deconvolution, there are also PEs

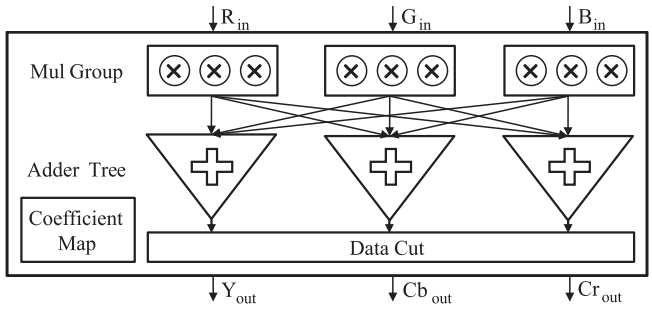

Fig. 3. Structure of RGB2YCbCr unit.

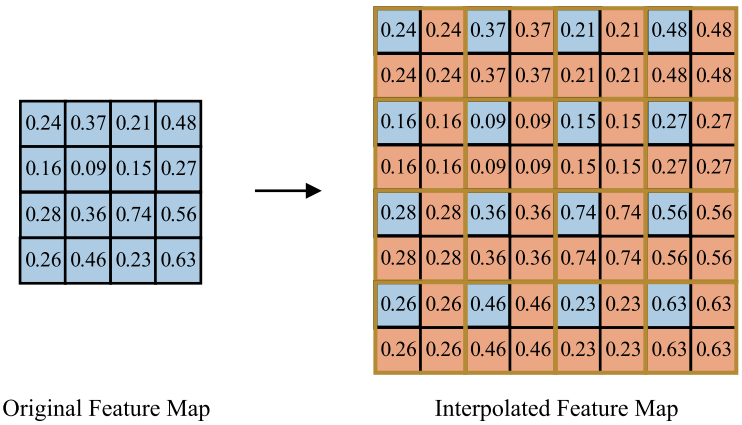

Fig. 4. Mapping of the nearest neighbor interpolation. The blue indicates the original data, and the orange indicates three copies of the original data.

for convolution, which are PE_1 (kernel size 1×1), PE_3 (kernel size 3×3), and PE_5 (kernel size 5×5). In order to make the computation resource distribution in balance, we divide the inference of the whole network into three parts and map them to corresponding PE cores to complete the computation. Among them, core 1 completes 5×5 and 1×1 convolution, corresponding to the first two layers of the network. Core 2 implements 3×3 convolution, corresponding to the middle four layers with 3×3 convolution. Core 3 accomplishes the operation of the last two layers, including 1×1 convolution and 9×9 deconvolution. The A/W Distributor maps input feature map data and weight data to the specific PE units, and the Shared Adder Tree unit completes the accumulation of PE results. The final output data from PE blocks are sent to the Output Reshaper unit for data alignment. Because the all-on-chip structure is adopted, the data generated between layers is stored in the Inter-layer Buffer.

3.2.3 Nearest Unit and T-buffer in Interpolation (Computation for Cb and Cr Channel). We adopt a simple interpolation scheme for the data from Cb and Cr channels, namely the nearest neighbor interpolation. According to the interpolation theory, we design a simple mapping scheme, that is, to make three copies of each data, and this method can be implemented without logical computation, as shown in Figure 4, so the resource overhead is low. T-buffer is used to store interpolated data temporarily.

3.2.4 YCbCr2RGB (Post Processing). The Y, Cb, and Cr component data obtained via PE cores and the nearest unit are sent to the YCbCr2RGB unit, converted to RGB color space, packaged,

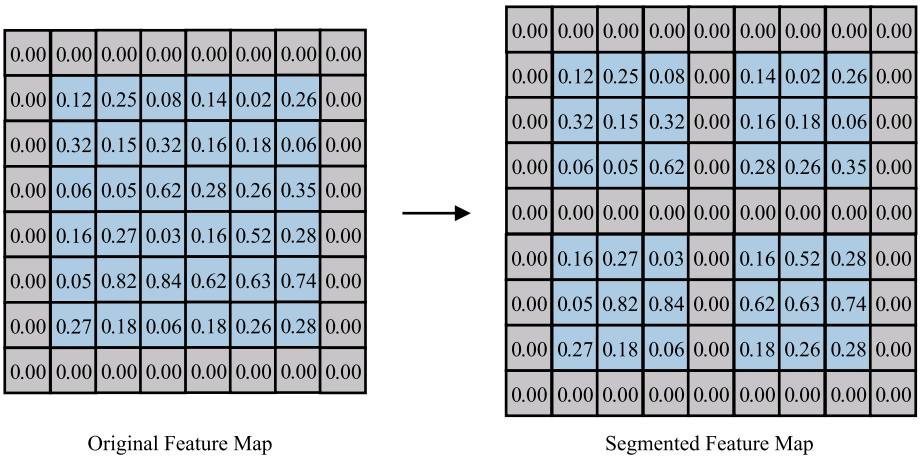

Fig. 5. An example of feature map segmentation. The blue indicates valid data, and the gray indicates padding data. An original feature map is segmented into four small features in this case.

and transferred to the off-chip memory through the DMA engine. The conversion operation and hardware implementation are similar to RGB2YCbCr, while the coefficients differ.

3.3 All-on-chip Mechanism

From observations (I) and (II), one of the significant super-resolution challenges is that the size of the inter-layer output feature map is very large, requiring a large storage capacity and frequent data access, which undoubtedly reduces the hardware efficiency. As shown in the last column of Table 1, if the output feature map data is directly stored in the inter-layer buffer, it is almost impossible to realize due to the limited on-chip memory resources of FPGA. Accordingly, it is an important issue how to relieve the pressure of memory access effectively.

3.3.1 Exploration. The feasible idea is to reduce the size of the input feature map so that the size of the output feature map is correspondingly shrunk, reducing the capacity required for on-chip storage and off-chip memory access. In other words, we can directly divide the input feature map into several small feature maps, send them to the network for inference, and then splice several small output feature maps as the final output. As shown in Figure 5, a significant feature map is segmented into four small feature maps and processed in turn. However, after the experimental evaluation, there is a noticeable decrease in PSNR and SSIM of the output feature maps. The main reason is that the segmentation operation destroys the spatial information features at the junction of small feature maps. Based on this problem, we refer to the block convolution method proposed by Li et al. [29], whose basic idea is to divide the original feature map into multiple blocks during training. In this way, it can eliminate the spatial correlation between blocks.

3.3.2 Procedure. Based on the above exploration, we modified the original training process. It mainly includes two differences: (1) we segment each input image in the training set into $N \times N$ (N is an even number) small feature maps and then send them to the network for retraining, and (2) in the experimental evaluation, the PSNR and SSIM metrics are evaluated after several small feature maps are spliced into a large feature map.

3.3.3 Results. We retrain the modified model and evaluate the PSNR and SSIM of the network when the parameter N takes different values. When the parameters N are set as 4, 8, 16, 24, 32, and

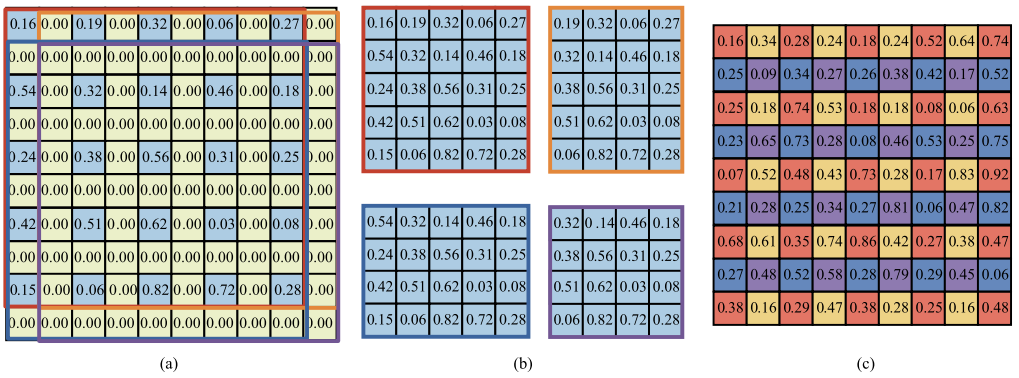

Fig. 6. Deconvolution optimization. (a) Interpolated input feature map; four boxes with different colors indicate four convolution inputs with size 9 × 9. (b) Four compressed convolution inputs with different sizes from (a). (c) Output feature map, where four colors represent that four outputs are achieved simultaneously.

40, respectively, we conduct retraining and evaluation on Set5, Set14, and BSD100 datasets. The results show that the PSNR and SSIM decreases are less than 0.02dB and 0.0005, respectively. The main reason is that the network size is small and the structure is simple, thus resulting in little impact on the final result after the segmentation operation. When the input image size is 1080p, we set the parameter N to 40. Compared with the traditional implementation, the feature map size is reduced by 1,600 times so that all the middle layer feature maps can be stored in the inter-chip buffer, significantly reducing the off-chip access and on-chip memory capacity.

4 HARDWARE-FRIENDLY AND EFFICIENT DECONVOLUTION DESIGN FOR *ADAS*

This section provides the problems and the optimization space of the deconvolution layer. Based on analyses, the efficient deconvolution scheme and hardware design are proposed.

4.1 Exploration of Efficient Deconvolution

As aforementioned, an optimization of the deconvolution layer is to remove the "0" value from the computation. In the following discussion, we employ scaling factor 2 (X2 SR) to analyze the computation efficiency of the deconvolution layer with the kernel size 9 × 9. Figure 6(a) is an interpolated input feature map; only 25 data in the red box are valid, and the remaining 69.1% of the data is a "0" value, which can be weeded out from the computation. As shown in Figure 6(b), we compress the "0" value to obtain valid input with size 5 × 5 marked in red, and the kernel size is also compressed to 5 × 5, realizing the convolution from 9 × 9 to 5 × 5. Using the above compression method and continuing to slide the convolution kernel, we can get a total of four inputs of different sizes, which are 5 × 5 (marked in red), 5 × 4 (marked in orange), 4 × 5 (marked in sky blue), and 4 × 4 (marked in purple), respectively.

4.1.1 Exploration of Computing Resource Utilization and Load Balance. As shown in Figure 6(a), the adjacent four inputs marked by different colors are calculated with corresponding weights to perform convolution operation. We observe that the required input data remains unchanged. Only the corresponding weights should be adjusted where the resource can be shared. In other words, no new input data is introduced for these four convolutions. In order to improve the parallelism of calculation, four convolutions with size 9 × 9 can be calculated simultaneously for the same interpolated input as shown in Figure 6(a). Figure 6(b) illustrates the compressed input corresponding to the four convolutions. Correspondingly, the PE with size 5 × 5 is usually used to meet the

requirements of four convolutions concurrently. However, for computing resources, there will be different degrees of resource waste for the input size smaller than 5×5. Take 5×4 convolution as an example: 20% of multiplication resources are not used. In general, 0%, 20%, 20%, and 36% of PE resources are wasted for the above four convolutions.

In addition, the PE may not be fully utilized when the input data size is smaller than the hard-coded kernel, causing a computation load to unbalance. Several solutions tend to solve the load unbalance issue. For example, the TDC method redistributes computing tasks to different PEs [7]. However, the load unbalance problem still influences the computation performance, and PE utilization can be further increased. Therefore, we develop a dynamic reconfigurable PE, recognized as DR_PE, which can simultaneously compute the full four convolutions. The PE utilization can reach nearly 100%.

4.1.2 Exploration of Data Alignment. Figure 6(c) shows the output feature map, where four different colors represent that four results are obtained simultaneously under the parallelism of 4. However, the upper and lower adjacent four output results are located in two different rows instead of output continuously in a row sequence, which should be re-arranged. We modify the output buffer and design a data reshaper to make the data output in sequence, only inducing a slight overhead.

4.2 *ADAS* Acceleration for Deconvolution Layer

Based on the above analyses, we develop a hardware structure to support the efficient acceleration of the deconvolution layer, as shown in Figure 7(a). The overall structure consists of an A/W data distributor, PE array, shared adder tree, output data truncation, and reshaper unit, which are also core components of *ADAS* architecture as shown in Figure 2. The A/W distributor realizes the mapping of the input feature map and weight and assigns them to different PEs to accomplish the computation. The PE array includes nine DR_PE units. Each DR_PE unit can complete nine MAC (multiplication and addition) operations. The shared adder tree unit accumulates the results of DR_PEs to obtain the intermediate results of convolutions. The output data is truncated in the data truncation unit and then sent to the output reshaper unit to realize data alignment and get the final output in sequence.

4.2.1 DR_PE Design. The structure of DR_PE is shown in Figure 7(a), which can realize the MAC operations of nine pairs of feature map data A_i and weight data W_i. The DR_PE can work on three computation modes controlled by two multiplexers (MUX). The details of the different modes are as follows:

Mode 0: output one convolution result with an input data length of 9.

Mode 1: output two convolution results with input data lengths of 7 and 2, respectively.

Mode 2: output three convolution results with input data lengths of 4, 3, and 2, respectively.

Based on the above three modes, the DR_PE can perform the up-sampling operation (i.e., deconvolution) at different scaling factors with only slightly additional logic resource overhead, where just two multiplexers are added for DR_PE compared with the traditional method.

4.2.2 Mapping Scheme. The integer SR scaling factor is primarily used in the practical application of the SR model. In addition, there are few super-resolution application scenarios based on fractional scaling factors, and its hardware implementation overhead is significant, resulting in extremely low computational efficiency. Therefore, based on the DR_PE unit, we discuss three commonly used scaling factors, 2, 3, and 4, respectively. We first introduce the mapping method at different scaling factors. Concurrently, the corresponding PE utilization and computation load are analyzed.

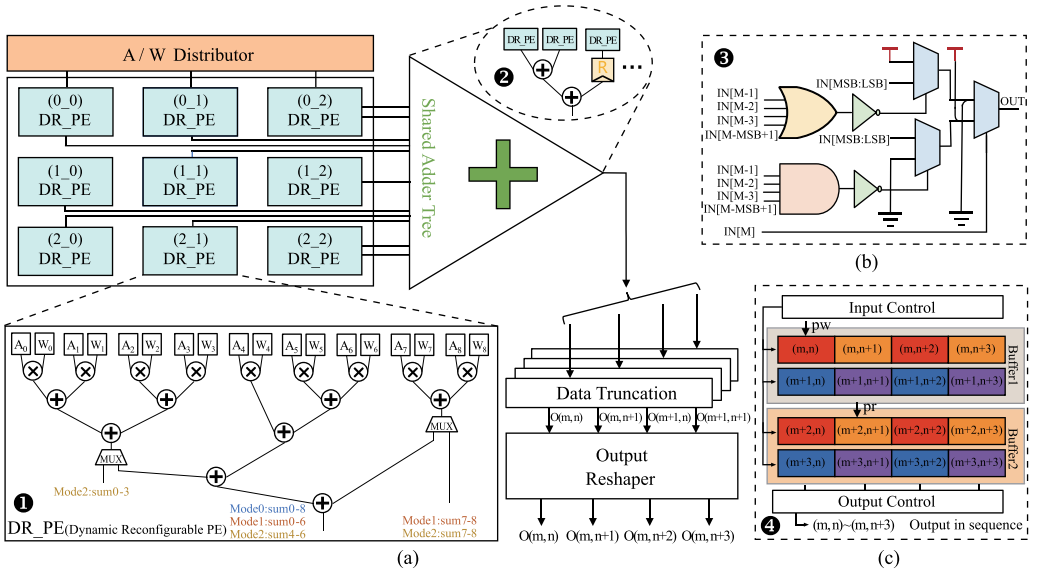

Fig. 7. Efficient deconvolution hardware structure for *ADAS*. (a) Proposed deconvolution hardware design and DR_PE structure. (b) Data truncation circuit. (c) Output Reshaper diagram. PW and pr represent the write pointer and read pointer, respectively. The operation process is ❶ → ❷ → ❸ → ❹.

Table 2. X2 SR Convolution Compression, Decomposition, and Mapping

Convolution Size after Compression	Convolution Length after Decomposition	Mapped DR_PE and Corresponding Working Mode
5 × 5	9 + 9 + 7	DR_PE(0_0) M0[a], DR_PE(0_1) M0, DR_PE(0_2) M1
5 × 4	9 + 9 + 2	DR_PE(1_0) M0, DR_PE(1_1) M0, DR_PE(0_2) M1
4 × 5	9 + 9 + 2	DR_PE(2_0) M0, DR_PE(2_1) M0, DR_PE(2_2) M1
4 × 4	9 + 7	DR_PE(1_2) M0, DR_PE(2_2) M1

[a]M0 and M1 represent mode 0 and mode 1, respectively.

X2 SR: If the scaling factor is 2, four deconvolutions can be computed simultaneously. Following the compression method introduced in Section 4.1 and for four convolutions with size 9 × 9, we can compress them to 5 × 5, 5 × 4, 4 × 5, and 4 × 4, respectively. Here, 81 MAC operations are required, consistent with the number of MACs supported by the PE array. To adapt the computation mode of DR_PE, we decompose and map the four compressed convolutions. The specific decomposition process is shown in column 2 of Table 2; 5 × 5 convolution can be decomposed into two convolutions with a length of 9 and one convolution with a length of 7. Both 5 × 4 and 4 × 5 convolutions can be decomposed into two convolutions with a length of 9 and one convolution with a length of 2. The 4 × 4 convolution can be decomposed into one convolution with a length of 9 and one convolution with a length of 7.

Next, we map the decomposed convolutions to the corresponding DR_PEs working in different modes, shown in column 3 of Table 2. The kernel size of convolution is 5 × 5. It is mapped to DR_PE numbered (0_0), (0_1), and (0_2), working in mode 0, mode 0, and mode 1, respectively. A more intuitive mapping can be seen in Figure 8(a), where seven DR_PEs work in mode 0 and two DR_PEs work in mode 1 in the PE array. In this case, every DR_PE in the PE array works with 100% utilization if not considering padding data. Correspondingly, there is no waste of computing resources, thus avoiding computation load unbalance.

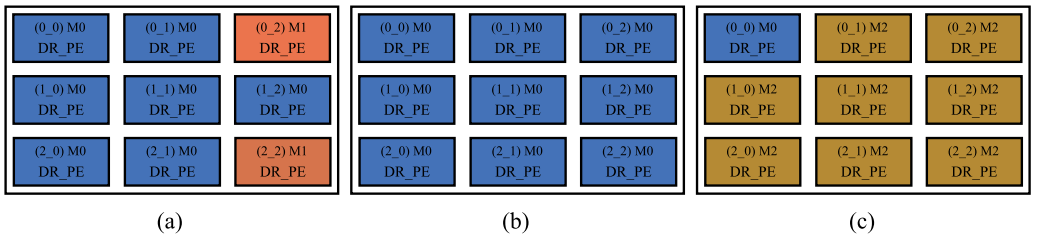

Fig. 8. DR_PE allocation. (a) PE array under the scaling factor 2. (b) PE array under the scaling factor 3. (c) PE array under the scaling factor 4. DR_PE marked in blue works in mode 0, DR_PE marked in orange works in mode one, and DR_PE marked in brown works in mode 2.

Table 3. X3 SR Convolution Compression, Decomposition, and Mapping

Convolution Size after Compression	Convolution Length after Decomposition	Mapped DR_PE and Corresponding Working Mode
3×3	9	DR_PE(0_0) M0
3×3	9	DR_PE(0_1) M0
3×3	9	DR_PE(0_2) M0
3×3	9	DR_PE(1_0) M0
3×3	9	DR_PE(1_1) M0
3×3	9	DR_PE(1_2) M0
3×3	9	DR_PE(2_0) M0
3×3	9	DR_PE(2_1) M0
3×3	9	DR_PE(2_2) M0

X3 SR: If the scaling factor is set to 3, nine convolution results can be output simultaneously. The input data of nine convolutions (upper and lower) are the same. Nine convolutions are compressed to obtain nine convolutions with size 3×3, which do not need to decompose. Corresponding convolution compression and DR_PE mapping are shown in Table 3. Compared with X2 SR, all compressed convolutions work in mode 0, and the computation is common. The intuitive corresponding DR_PE allocation in the PE array is shown in Figure 8(b). In this case, every DR_PE in the PE array also works with 100% utilization if not considering padding data, thus avoiding computation load unbalance.

X4 SR: If the scaling factor is 4, 16 convolution results can synchronously be obtained. The input data of 16 convolutions (upper and lower) remain unchanged. Compress the input data, and the corresponding 16 compressed convolutions and mappings are shown in Table 4. It can be found that except for one DR_PE that works in mode 0, the others work in mode 2. The specific PE array allocation is shown in Figure 8(c). In this case, 100% utilization of each DR_PE is also obtained without considering padding data.

The mapping method can be predetermined for the above three scaling factors because the specific deconvolution parameters are known. With our DR_PE unit, different scaling-factor deconvolutions can be realized, which supports dynamic reconfiguration. The hardware implementation is efficient without additional computing resource overhead (i.e., multiplier and adder). Under different scaling factors, PE utilization is nearly 100%, alleviating the load imbalance problem.

4.2.3 Data Truncation. For the fixed-point results output by convolution operation, data truncation is required to keep the input and output data widths consistent. The specific circuit is shown in Figure 7(b), which mainly includes the sign decision part and overflow decision part. First,

Table 4. X4 SR Convolution Compression, Decomposition, and Mapping

Convolution Size after Compression	Convolution Length after Decomposition	Mapped DR_PE and Corresponding Working Mode
3 × 3	9	DR_PE(0_0) M0
3 × 2	3 + 3	DR_PE(0_1) M2, DR_PE(0_2) M2
3 × 2	3 + 3	DR_PE(1_0) M2, DR_PE(1_1) M2
3 × 2	3 + 3	DR_PE(1_2) M2, DR_PE(2_0) M2
2 × 3	3 + 3	DR_PE(2_1) M2, DR_PE(2_2) M2
2 × 2	4	DR_PE(0_1) M2
2 × 2	4	DR_PE(0_2) M2
2 × 2	4	DR_PE(1_0) M2
2 × 3	2 + 2 + 2	DR_PE(1_0) M2, DR_PE(1_1) M2, DR_PE(1_2) M2
2 × 2	4	DR_PE(1_1) M2
2 × 2	4	DR_PE(1_2) M2
2 × 2	4	DR_PE(2_0) M2
2 × 3	2 + 2 + 2	DR_PE(2_0) M2, DR_PE(2_1) M2, DR_PE(2_2) M2
2 × 2	4	DR_PE(2_1) M2
2 × 2	4	DR_PE(2_2) M2
2 × 2	2 + 2	DR_PE(0_1) M2, DR_PE(0_2) M2

determine the positive or negative of the data. M represents the highest bit of the input data IN. If IN[M] = 1, the data is negative, and the lower half of the circuit is activated. Inversely, the upper half of the circuit is activated. Second, determine the overflow of the data. Typically, the truncated data may be from the **most significant bits (MSBs)** to the **least significant bits (LSBs)**. Take positive data as an example. Skipping the sign bit, the input data from the highest bit to the MSB executes the NOR operation bit by bit. If the output is 0, there is no overflow, and the truncated data is directly output, that is, IN[MSB:LSB]. On the contrary, overflow occurs, and the maximum value is output. The overflow decision of negative data is similar; the only difference is that the NAND operation carries out the logic operation.

4.2.4 Data Alignment. As mentioned in Section 4.1, the final output is not in sequence. Therefore, we implement data alignment based on the output buffer. As shown in Figure 7(c), we use two buffers, each of which can store two lines of data, and two buffer pointers are set to represent the read pointer and write pointer, respectively. Take X2 SR, for example, where each operation can output four data simultaneously. Initially, the write pointer points to buffer 1. For the four final output data, the first two data are stored in line 1 of buffer 1, and the last two ones are in line 2 of buffer 1. Repeat the above operation until buffer 1 is full. Then, the write pointer points to buffer two while the read pointer points to buffer one and outputs the final results in sequence. Compared with the traditional output buffer, it not only introduces almost no control resource overhead but also realizes the sequential output.

5 EXPERIMENT AND EVALUATION

This section introduces the experiment setup, including network training, hardware configuration, and network quantization. Also, we discuss evaluation results on image quality and hardware performance. Then, the analyses and future work are presented.

5.1 Experiment Setup

The experiment setup contains network training for the hardware optimization and hardware configuration for FPGA implementation.

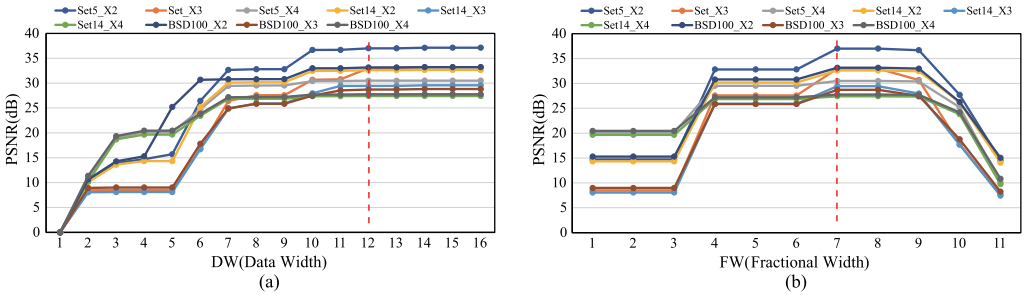

Fig. 9. Network quantization. (a) The optimal PSNR under different DWs. (b) When DW is 12 and FW alters, the changes in PSNR.

5.1.1 Network Training. Before training, data enhancement is implemented for initial training images, such as rotation and scaling, to ensure that more helpful image information is obtained during training. Inspired by [29], we modify the training process to retrain the network. In the training process, several tricks are applied. First, we segment each training set image into multiple small images to reduce memory resources on-chip. The image reconstruction quality can be maintained with our all-on-chip mechanism. Second, we change the optimizer to Adam, which can achieve a better training effect through various experimental tests.

5.1.2 Hardware Configuration. In the hardware deployment and evaluation, the Xilinx Vivado 2019.2 development environment and Verilog HDL are used to realize the hardware SR architecture *ADAS*, and Xilinx VC707 Evaluation Board is selected for implementation and analyses.

5.2 Network Quantization

We quantize the feature map and weight data from the 32-bit floating points to the fixed points. To begin with, we represent the fixed-point data by **data width (DW)**, composed of [S, IW, FW], indicating the sign bit, integer bits, and fractional bits, respectively. We select Set5, Set14, and BSD100 datasets and quantize them at different scaling factors. The specific process consists of two steps. Step 1 and for DW with a given width: we traverse the corresponding PSNR when IW and FW are in different bit widths. Then, the highest PSNR is selected. Next, we change the width of DW in turn and repeat the above operations to find the optimal DW. Figure 9(a) shows the corresponding maximum PSNR under different DWs. It can be seen that the PSNR inclines with the increase of DW. When it reaches 12 bits, the PSNR remains stable. Therefore, we use the fixed-point data when DW is 12 bits. Step 2 and to search the corresponding IW and FW: Figure 9(b) indicates the relationship among FW, data accuracy, and quantization error. When FW increases, the data accuracy increases, corresponding quantization error decreases, and PSNR gradually increases. The PSNR reaches the maximum while the FW becomes 7.

Based on the above analyses, we determine the DW of quantized fixed-point data to 12 bits, and the specific combination is [1, 4, 7]. Without quantization, the *ADAS* software optimized version (*ADAS*-SW) can achieve a PSNR of 37.15dB, higher than the original FSRCNN network [12], as shown in Table 5. Based on the proposed quantization method, the PSNR of the *ADAS* entirely hardware (*ADAS*-FH) can reach 37.08dB under the scaling factor of 2 and dataset Set5.

5.3 Image Quality Evaluation

5.3.1 PSNR Comparison with Different Scaling Factors. For comparison, we choose the Set5, Set14, and BSD100 datasets for the evaluation. Specifically, the PSNRs of *ADAS*-SW and

Table 5. PSNR Comparison of Image Quality

PSNR (dB)	X2			X3			X4		
	Set5[a]	Set14[b]	BSD100[c]	Set5	Set14	BSD100	Set5	Set14	BSD100
FSRCNN [12] (Baseline)	37.00	32.63	31.53	33.16	29.43	28.53	30.71	27.59	26.98
ADAS-SW (32b)	37.15	32.68	33.35	33.25	29.58	28.81	30.66	27.62	27.76
ADAS-FH (12b)	37.08	32.63	33.20	33.14	29.45	28.78	30.64	27.61	27.75
Accuracy comparison (*ADAS*-SW vs. Baseline)	**+0.15**	**+0.05**	**+1.82**	**+0.09**	**+0.15**	**+0.28**	-0.05	**+0.03**	**+0.78**

[a]Set5 contains five images for the test.
[b]Set14 contains 14 images for the test.
[c]BSD100 contains 100 images for the test.

Fig. 10. Visual quality comparison of different SR networks with the scaling factor 2.

ADAS-FH are tested under the scaling factor of X2, X3, and X4, respectively. Table 5 demonstrates the comparison results. The PSNR of *ADAS*-SW is higher than the original FSRCNN network with scaling factors 2 and 3 for the effect tricks applied in our *ADAS*-SW training. When the scaling factor is 4, only the PSNR of the case for the Set5 dataset is lower than the original network. The primary reason is that the Set5 dataset lacks sample images and image information. On the contrary, the BSD100 dataset contains more realistic images, and the sample size is large enough to get good image quality.

Figure 10 illustrates the experimental visual quality of *ADAS* under the Set5 dataset. With the X2 scaling factor, we test the SR networks such as Bicubic, SRCNN, FSRCNN, *ADAS*-SW, and *ADAS*-FH. As a result, both *ADAS*-SW and *ADAS*-FH can achieve much better image quality in terms of processing texture and edge details.

5.3.2 SSIM Comparison with Different Scaling Factors. Compared with PSNR, SSIM can better reflect the subjective feeling of human eyes by measuring image similarity from luminance, contrast, and structure. The test results are shown in Table 6, where the SSIM of the *ADAS* version is better than other counterparts in most cases. Only the case of X4 SR under the Set5 dataset is lower than that of the original network. For the BSD100 dataset, the SSIM of our proposed *ADAS* can achieve significant improvement.

5.4 Hardware Metrics Comparison and Analyses

The target of *ADAS* is hardware acceleration with high image quality. Here, we discuss the hardware-based image quality, memory access, and performance to validate the efficiency of our proposed techniques. Fair comparisons are also demonstrated in our contributions and the novelty of this work.

Table 6. SSIM Comparison of Image Quality

SSIM	X2			X3			X4		
	Set5	Set14	BSD100	Set5	Set14	BSD100	Set5	Set14	BSD100
FSRCNN [12] (Baseline)	0.9558	0.9088	0.8920	0.9140	0.8242	0.7910	0.8657	0.7535	0.7150
ADAS-SW (32b)	0.9587	0.9134	0.9206	0.9226	0.8405	0.8043	0.8644	0.7612	0.7465
ADAS-FH (12b)	0.9586	0.9132	0.9200	0.9223	0.8401	0.8039	0.8643	0.7610	0.7463
Accuracy comparison (*ADAS*-SW vs. Baseline)	**+0.0029**	**+0.0046**	**+0.0286**	**+0.0086**	**+0.0163**	**+0.0133**	-0.0013	**+0.0077**	**+0.0315**

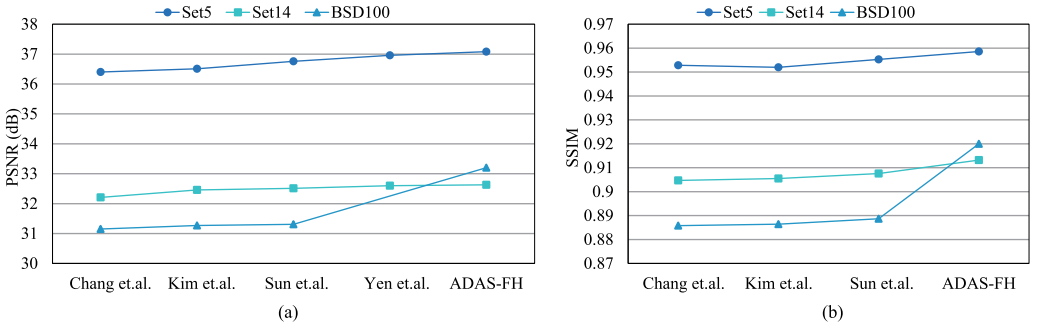

Fig. 11. Comparison of hardware-based image quality. (a) PSNR comparison. (b) SSIM comparison. *ADAS* is compared to Chang et al. [6], Kim et al. [24], Sun et al. [32], and Yen et al. [40], respectively. Note that the work of Yen et al. [40] has no experimental data for the BSD100 dataset.

5.4.1 Hardware-based Image Quality Analyses. Figure 11 demonstrates the comparison of image quality, including both PSNR and SSIM, obtained by different SR-based hardware on the Set5, Set14, and BSD100 datasets, respectively. We notice that several compared hardware works only support the scaling factor of 2. We indicate X2 SR *ADAS* in the comparison. Based on the image quality comparison, our proposed *ADAS*-FH obtains the best results in terms of PSNR and SSIM metrics. The main reason is that we optimized the original FSRCNN considering both the algorithm's trimming and our hardware deployment target. We optimize the algorithm to try various retraining parameters and analyze the best quantization condition.

5.4.2 Memory Access Analyses. Figure 12(a) provides the comparison of on-chip and off-chip memory access between *ADAS*-FH and baseline hardware without optimization. Our proposed *ADAS* can reduce memory accesses by 99.28%, 98.57%, and 97.59% with scaling factors X2, X3, and X4 SR, respectively. Based on our proposed all-on-chip *ADAS* architecture, the output feature maps of middle layers are stored on the chip with small-size memory. Only the first and last layer feature maps need to interact with off-chip memory, which reduces the memory access between on-chip and off-chip, alleviates the pressure of bandwidth, and ensures real-time performance.

Figure 12(b) compares memory access with previous SR hardware. *ADAS*-FH can achieve the maximum bandwidth reduction of 97% and obtains a maximum bandwidth improvement of 1.23×. In addition, compared with the state of the art on bandwidth optimization, *ADAS*-FH can still achieve nearly 3% bandwidth improvement.

5.4.3 Performance Analyses. *ADAS* can provide high performance based on our developed deconvolution scheme and all-on-chip architecture. Figure 13 compares our *ADAS*-FH with baseline and several great SR hardware works. The latency represents the inference time speedup of hardware after adopting different optimization strategies. Compared to the non-optimized hardware baseline, the end-to-end latency of the proposed hardware architecture *ADAS*-FH can be improved

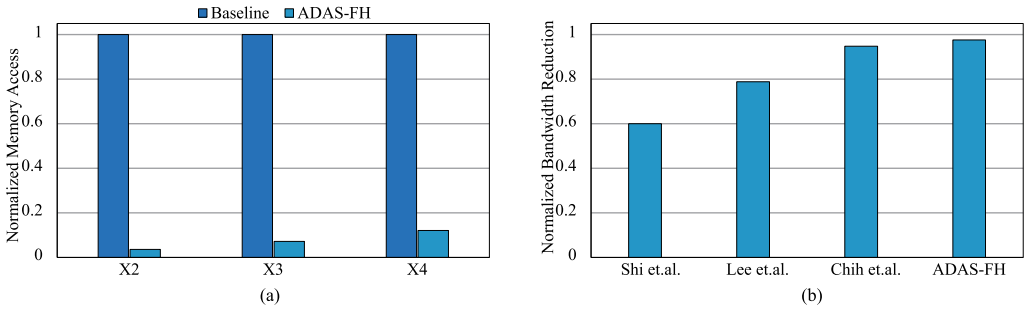

Fig. 12. Comparison of memory access. (a) Memory access comparison. (b) Bandwidth reduction comparison. The above three works, Shi et al., Lee et al., and Chih et al., are from references [31], [26], and [8], respectively.

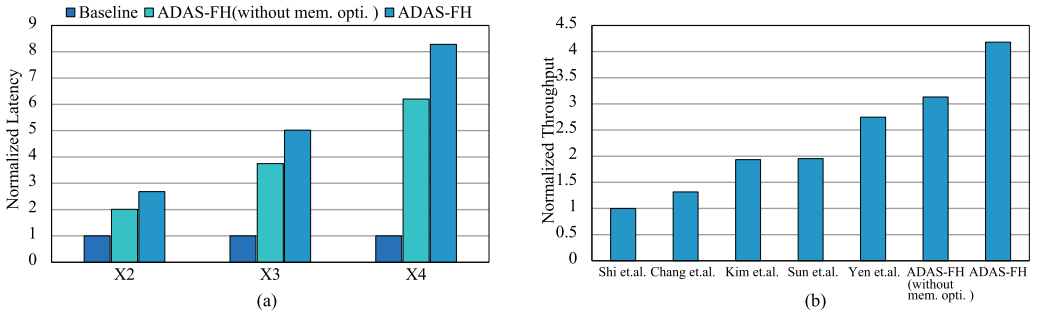

Fig. 13. Comparison of performance. (a) Latency comparison. Without mem. opti. indicates *ADAS*-FH does not adopt a memory optimization mechanism. (b) Performance comparison. The above five works, Shi et al., Chang et al., Kim et al., Sun et al., and Yen et al., are from references [31], [6], [24], [32] and [40], respectively.

by 2.68×, 5.02×, and 8.28×, respectively, in the case of X2, X3, and X4 SR, as shown in Figure 13(a). The improvements are mainly from two aspects. On the one hand, the hardware implementation of the deconvolution layer reduces invalid computation and thus improves hardware efficiency. The convolution compression technique is employed to skip the invalid computation with the value "0", which reduces nearly 75% of the MAC operation of the deconvolution layer. In addition, the DR_PE technique allocates computing resources to improve hardware utilization, reducing end-to-end latency. In this way, *ADAS*-FH can achieve latency improvement of 2.01×, 3.75×, and 6.20×, respectively, in the case of X2, X3, and X4 SR, which can be seen in the column *ADAS*-FH (without mem. opti.) of Figure 13(a).

On the other hand, the proposed all-on-chip architecture significantly reduces memory access between on-chip and off-chip as described in the above analyses, which further reduces the latency, as shown in the column *ADAS*-FH of Figure 13(a). In summary, *ADAS*-FH achieves a good acceleration efficiency based on the breakdown of the latency improvement, where the deconvolution optimization scheme provides 75% latency improvement. In order to evaluate the performance of our architecture *ADAS*, we compared our work with previous SR hardware. As shown in Figure 13(b), the proposed architecture *ADAS*-FH can achieve a maximum performance improvement of 4.18×. Even if the all-on-chip architecture based on memory access optimization is not adopted, *ADAS*-FH can still achieve a performance improvement of 3.13×. Further, compared with the state-of-the-art work of Yen et al. [40], our architecture can achieve a performance improvement of 1.52×.

5.4.4 Architecture Overall Comparison. Table 7 demonstrates the comparison of different hardware-based SR architectures in recent years. Our proposed architecture *ADAS* can support

Table 7. Comparison of Different Hardware Architectures

From	Chang et al. [6]	Kim et al. [24]	Shi et al. [31]	Sun et al. [32]	Yen et al. [40]	*ADAS*-FH (Proposed)
SR Methods	FSRCNN-based	CNN-based	FSRCNN-based	RNN-based	IDN-based	FSRCNN-based
Supported Scales	2, 3, 4	2	2	2	2	2, 3, 4
Technology	Xilinx XC7K410T	Xilinx XCKU040	Xilinx ZCU102	Xilinx XCKU15P	SYNOPSYS 32nm EDK	Xilinx XC7VX485T
Frequency (MHz)	130	150	200	160	200	200
Precision	13 bit	10-14 bit	16 bit	12-16 bit	12 bit	12 bit
HW Resources[a]	LUT:167K, REG:158K, DSP:1,512	LUT:151K, REG:121K, DSP:1,920	LUT:173K, REG:-, DSP:746	LUT:98K, REG:57K, DSP:1,820	3,114K	LUT:105K, REG:123K, DSP:1,558
Memory Size (KB)	945	194	1,396	4,842	-	1,118
Target Resolution	4K UHD	4K UHD	4K UHD	4K UHD	1080P FHD	4K UHD
Frame Rate (fps)	62.7	60	120.4	76	60	96.4

[a] A two-input NAND gate is counted as one equivalent gate.

three different scaling factors, X2, X3, and X4, while most other SR architectures only support one scaling factor, X2. Concerning logical resource overhead, the lookup table and register resources occupation are much lower than the other two hardware systems based on the FSRCNN network. And the two networks based on FSRCNN have been trimmed [6, 31]. However, since we do not trim the original FSRCNN network, the network size is larger than the other two. In order to ensure performance, DSP occupies more than [31]. Specifically, the amount of computation required in *ADAS* is 54.1*GOPS*, while the network used in [31] is a lightweight FSRCNN-s, which only requires 24.3*GOPS*, 55.1% less than the network FSRCNN used by *ADAS*. In addition, [31] adopts the Linear Total Variation Algorithm, which computes the linear **Total Variation (TV)** value of small image blocks. If it is higher than the threshold value, the image blocks will be sent to the neural network FSRCNN-s to complete computation; otherwise, traditional interpolation algorithms will be used. This way, the computation can be further reduced, but the PSNR is degraded. In order to highlight the advantages of *ADAS*, the FSRCNN-s used in [31] is implemented with *ADAS*, and finally can reach nearly $305fps$, which is 2.5× that of [31].

Due to the employment of a feature map segmentation strategy, the memory size is small for an all-on-chip architecture. From Section 5.4.1, *ADAS*-FH can achieve the highest PSNR and SSIM in the image quality evaluation. In addition, high performance is obtained from Section 5.4.3, and *ADAS*-FH can achieve a frame rate of $96.4fps$, which has higher real-time performance than other architectures.

5.5 Discussion

5.5.1 Tradeoff Analyses. In our proposed hardware *ADAS* architecture, due to the dynamically reconfigurable characteristic of DR_PE, we can dynamically obtain the output of different scaling factors with the same PE array. Here, the smaller scaling factor can get a better SR result. Thus, a small scaling factor is usually adopted. For 4K output, we can use 1080p as input and realize 4K high-resolution output based on X2 SR. For 8K output, there is a tradeoff between speed and accuracy. On the one hand, to pursue high image quality, we can continue to set the scaling factor as 2. In this case, the input size is 4K, and it will lead to more computation. On the other hand, if X4 SR is used, the input becomes 1080p, which can achieve faster speed, but the image quality could be better than the former due to the degraded accuracy. According to the practical application requirements, we can adopt different SR methods.

5.5.2 Future Work. The proposed *ADAS* architecture can achieve 100% utilization under the deconvolution layer of the FSRCNN model. We hope the *ADAS* architecture can support different

Table 8. 7 × 7 Deconvolution Compression, Decomposition, and Mapping

Convolution Size after Compression	Convolution Length after Decomposition	Mapped DR_PE and Corresponding Working Mode
4 × 4	4 + 4 + 4 + 4	DR_PE(0_0) M2, DR_PE(0_1) M2, DR_PE(0_2) M2, DR_PE(1_0) M2
4 × 3	3 + 3 + 3 + 3	DR_PE(0_0) M2, DR_PE(0_1) M2, DR_PE(0_2) M2, DR_PE(1_0) M2
3 × 4	2 + 2 + 2 + 2 + 2 + 2	DR_PE(0_0) M2, DR_PE(0_1) M2, DR_PE(0_2) M2, DR_PE(1_0) M2, DR_PE(1_1) M2, DR_PE(1_2) M2
3 × 3	9	DR_PE(2_2) M0
4 × 4	4 + 4 + 4 + 4	DR_PE(1_1) M2, DR_PE(1_2) M2, DR_PE(2_0) M2, DR_PE(2_1) M2
4 × 3	3 + 3 + 3 + 3	DR_PE(1_1) M2, DR_PE(1_2) M2, DR_PE(2_0) M2, DR_PE(2_1) M2

deconvolution kernel sizes. In addition, how to support more super-resolution models to improve the versatility of *ADAS* architecture needs further exploration in the future.

The specific exploration is as follows:

- To explore the universality of the deconvolution scheme. Our *ADAS* also can support other deconvolution kernel sizes beyond the kernel size 9 × 9. For example, set the deconvolution size as 7 × 7, and compress it using the proposed method. In the case of X2 SR, a total of four convolutions of different sizes can be obtained, which are 4 × 4, 4 × 3, 3 × 4, and 3 × 3. In order to improve the computational utilization, the DR_PE array can compute six compressed convolutions at the same time, which are 4 × 4, 4 × 3, 3 × 4, 3 × 3, 4 × 4, and 4 × 3, and the specific allocation and mapping are shown in Table 8. The average utilization can reach 95.1%, and the speedup is 3.63 compared with the baseline.
  Further, we need to design a general mapping algorithm. In this way, we can use automatic tools to search for a mapping scheme with the highest computational utilization for any deconvolution of different sizes.
- Exploration on the universality of *ADAS* architecture. In order to support different super-resolution models, we should adopt more operators in the *ADAS* architecture. For example, we need to add the implementation of operations such as concatenation and slice for the IDN model. In addition, the data flow control needs to be further designed to switch to the computation of different super-resolution models efficiently.
- To explore the all-on-chip mechanism. Although our all-on-chip mechanism has achieved good results in the FSRCNN network, when the network structure becomes more complex, we need to guarantee the same image quality. In addition, compared with reference [29], which method is more robust and versatile also needs further comparison.

6 CONCLUSION

In this article, we proposed an ADAS hardware accelerator with a hardware-efficient deconvolution scheme, which significantly reduces the amount of computation required for inference. Also, the ADAS improves PE utilization and eliminates computing load unbalances. Furthermore, the ADAS significantly alleviates the memory wall problem caused by the bandwidth limitation. ADAS can achieve better reconstruction performance through the FPGA-based implementation and a series of experimental analyses. Besides, the significant speedup is achieved under different scaling factors. Concurrently, it can dynamically support various scaling factors to produce 4K and 8K ultra-high resolution. We will explore the optimization space for super-resolution in hardware deployment in future work.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions.

REFERENCES

[1] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. IEEE, 1–12.

[2] Prasoon Ambalathankandy, Shinya Takamaeda, Motomura Masato, Tetsuya Asai, Masayuki Ikebe, and Hotaka Kusano. 2018. Real-time HDTV to 4K and 8K-UHD conversions using anti-aliasing based super resolution algorithm on FPGA. *Microprocessors and Microsystems* 61 (2018), 21–31.

[3] Saeed Anwar and Nick Barnes. 2020. Densely residual Laplacian super-resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 3 (2020), 1192–1124.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934* x, x (2020).

[5] Kelvin C. K. Chan, Shangchen Zhou, Xiangyu Xu, and Chen Change Loy. 2022. Investigating tradeoffs in real-world video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVF, 5962–5971.

[6] Jung-Woo Chang, Keon-Woo Kang, and Suk-Ju Kang. 2018. An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution. *IEEE Transactions on Circuits and Systems for Video Technology* 30, 1 (2018), 281–295.

[7] Jung-Woo Chang and Suk-Ju Kang. 2018. Optimizing FPGA-based convolutional neural networks accelerator for image super-resolution. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC'18)*. IEEE, 343–348.

[8] Chung-Yan Chih, Sih-Sian Wu, Jan P. Klopp, and Liang-Gee Chen. 2018. Accurate and bandwidth efficient architecture for CNN-based full-HD super-resolution. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS'18)*. IEEE, 1–5.

[9] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, and Qingyuan Li. 2021. Fast, accurate and lightweight super-resolution with neural architecture search. In *2020 25th International Conference on Pattern Recognition (ICPR'21)*. IEEE, 59–64.

[10] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. 2019. Second-order attention network for single image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVF, 11065–11074.

[11] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38, 2 (2015), 295–307.

[12] Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the super-resolution convolutional neural network. In *European Conference on Computer Vision*. Springer, 391–407.

[13] Xinbo Gao, Kaibing Zhang, Dacheng Tao, and Xuelong Li. 2012. Image super-resolution with sparse neighbor embedding. *IEEE Transactions on Image Processing* 21, 7 (2012), 3194–3205.

[14] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and Jian Sun. 2021. OTA: Optimal transport assignment for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVF, 303–312.

[15] Zhuolun He, Hanxian Huang, Ming Jiang, Yuanchao Bai, and Guojie Luo. 2018. FPGA-based real-time super-resolution system for ultra high definition videos. In *2018 IEEE 26th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'18)*. IEEE, 181–188.

[16] Shuying Huang, Jun Sun, Yong Yang, Yuming Fang, Pan Lin, and Yue Que. 2018. Robust single-image super-resolution based on adaptive edge-preserving smoothing regularization. *IEEE Transactions on Image Processing* 27, 6 (2018), 2650–2663.

[17] Zheng Hui, Xiumei Wang, and Xinbo Gao. 2018. Fast and accurate single image super-resolution via information distillation network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. CVF, 723–731.

[18] Takashi Isobe, Xu Jia, Xin Tao, Changlin Li, Ruihuang Li, Yongjie Shi, Jing Mu, Huchuan Lu, and Yu-Wing Tai. 2022. Look back and forth: Video super-resolution with explicit temporal difference modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVF, 17411–17420.

[19] Takashi Isobe, Songjiang Li, Xu Jia, Shanxin Yuan, Gregory Slabaugh, Chunjing Xu, Ya-Li Li, Shengjin Wang, and Qi Tian. 2020. Video super-resolution with temporal group attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVF, 8008–8017.

[20] Ildoo Kim, Woonhyuk Baek, and Sungwoong Kim. 2020. Spatially attentive output layer for image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVF, 9533–9542.

[21] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Deeply-recursive convolutional network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. CVF, 1637–1645.

[22] Min Beom Kim, Sanglyn Lee, Ilho Kim, Hee Jung Hong, Chang Gone Kim, and Soo Young Yoon. 2020. Efficient multiquality super-resolution using a deep convolutional neural network for an FPGA implementation. *Journal of the Society for Information Display* 28, 5 (2020), 428–439.

[23] Yongwoo Kim, Jae-Seok Choi, and Munchurl Kim. 2018. 2X super-resolution hardware using edge-orientation-based linear mapping for real-time 4K UHD 60 fps video applications. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 9 (2018), 1274–1278.

[24] Yongwoo Kim, Jae-Seok Choi, and Munchurl Kim. 2018. A real-time convolutional neural network for super-resolution on FPGA with applications to 4K UHD 60 fps video services. *IEEE Transactions on Circuits and Systems for Video Technology* 29, 8 (2018), 2521–2534.

[25] Yongwoo Kim, Jae-Seok Choi, Jaehyup Lee, and Munchurl Kim. 2020. A CNN-based multi-scale super-resolution architecture on FPGA for 4K/8K UHD applications. In *International Conference on Multimedia Modeling*. Springer, 739–744.

[26] Juhyoung Lee, Jinsu Lee, and Hoi-Jun Yoo. 2020. SRNPU: An energy-efficient CNN-based super-resolution processor with tile-based selective super-resolution in mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10, 3 (2020), 320–334.

[27] Jooseung Lee and In-Cheol Park. 2016. High-performance low-area video up-scaling architecture for 4-K UHD video. *IEEE Transactions on Circuits and Systems II: Express Briefs* 64, 4 (2016), 437–441.

[28] Juhyoung Lee, Dongjoo Shin, Jinsu Lee, Jinmook Lee, Sanghoon Kang, and Hoi-Jun Yoo. 2019. A full HD 60 fps CNN super resolution processor with selective caching based layer fusion for mobile devices. In *2019 Symposium on VLSI Circuits*. IEEE, C302–C303.

[29] Gang Li, Zejian Liu, Fanrong Li, and Jian Cheng. 2022. Block convolution: Toward memory-efficient inference of large-scale CNNs on FPGA. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* 41, 5 (2022), 1436–1447.

[30] Wendong Mao, Jun Lin, and Zhongfeng Wang. 2020. F-DNA: Fast convolution architecture for deconvolutional network acceleration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 8 (2020), 1867–1880.

[31] Bizhao Shi, Zhucheng Tang, Guojie Luo, and Ming Jiang. 2019. Winograd-based real-time super-resolution system on FPGA. In *2019 International Conference on Field-Programmable Technology (ICFPT'19)*. IEEE, 423–426.

[32] Kaicong Sun, Maurice Koch, Zhe Wang, Slavisa Jovanovic, Hassan Rabah, and Sven Simon. 2022. An FPGA-based residual recurrent neural network for real-time video super-resolution. *IEEE Transactions on Circuits and Systems for Video Technology* 32, 4 (2022), 1739–1750.

[33] Rui Tang, Jonathon Dore, Jin Ma, and Philip H. W. Leong. 2021. Interpolating high granularity solar generation and load consumption data using super resolution generative adversarial network. *Applied Energy* 299 (2021), 117297.

[34] Zhihao Wang, Jian Chen, and Steven C. H. Hoi. 2021. Deep learning for image super-resolution: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 10 (2021), 3365–3387.

[35] Zhangyang Wang, Yingzhen Yang, Zhaowen Wang, Shiyu Chang, Jianchao Yang, and Thomas S. Huang. 2015. Learning super-resolution jointly from external and internal examples. *IEEE Transactions on Image Processing* 24, 11 (2015), 4359–4371.

[36] Yuzhuo Wei, Li Chen, Rong Xie, Li Song, Xiaoyun Zhang, and Zhiyong Gao. 2019. FPGA based video transcoding system with 2K-4K super-resolution conversion. In *2019 IEEE Visual Communications and Image Processing (VCIP'19)*. IEEE, 1–2.

[37] Xiaoyu Xiang, Yapeng Tian, Yulun Zhang, Yun Fu, Jan P. Allebach, and Chenliang Xu. 2020. Zooming slow-mo: Fast and accurate one-stage space-time video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVF, 3370–3379.

[38] Chih-Yuan Yang, Chao Ma, and Ming-Hsuan Yang. 2014. Single-image super-resolution: A benchmark. In *European Conference on Computer Vision*. Springer, 372–386.

[39] Ming-Che Yang, Kuan-Ling Liu, and Shao-Yi Chien. 2017. A real-time FHD learning-based super-resolution system without a frame buffer. *IEEE Transactions on Circuits and Systems II: Express Briefs* 64, 12 (2017), 1407–1411.

[40] Po-Wei Yen, Yu-Sheng Lin, Chia-Yang Chang, and Shao-Yi Chien. 2020. Real-time super resolution CNN accelerator with constant kernel size Winograd convolution. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS'20)*. IEEE, 193–197.

[41] Feng Yu, Yanpeng Cao, and Yongming Tang. 2020. Realization of quantized neural network for super-resolution on PYNQ. In *2020 IEEE 28th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'20)*. IEEE, 233–233.

[42] Yunxuan Yu, Tiandong Zhao, Mingyu Wang, Kun Wang, and Lei He. 2020. Uni-OPU: An FPGA-based uniform accelerator for convolutional and transposed convolutional networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 7 (2020), 1545–1556.

[43] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. 2018. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. CVF, 286–301.

[44] Yajia Zhang, Hongyi Sun, Jinyun Zhou, Jiacheng Pan, Jiangtao Hu, and Jinghao Miao. 2020. Optimal vehicle path planning using quadratic optimization for Baidu Apollo open platform. In *2020 IEEE Intelligent Vehicles Symposium (IV'20)*. IEEE, 978–984.

[45] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. 2018. Residual dense network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. CVF, 2472–2481.

[46] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*. CVF, 2223–2232.

Received 16 June 2022; revised 27 October 2022; accepted 2 November 2022