A CNN Inference Accelerator on FPGA With Compression and Layer-Chaining Techniques for Style Transfer Applications

Suchang Kim, *Graduate Student Member, IEEE*, Boseon Jang, *Graduate Student Member, IEEE*, Jaeyoung Lee, *Graduate Student Member, IEEE*, Hyungjoon Bae, *Graduate Student Member, IEEE*, Hyejung Jang, *Graduate Student Member, IEEE*, and In-Cheol Park, *Senior Member, IEEE*

***Abstract*— Recently, convolutional neural networks (CNNs) have actively been applied to computer vision applications such as style transfer that changes the style of a content image into that of a style image. As the style transfer CNNs are based on encoder-decoder network architecture and should deal with high-resolution images that become mainstream these days, the computational complexity and the feature map size are very large, preventing the CNNs from being implemented on an FPGA. This paper proposes a CNN inference accelerator for the style transfer applications, which employs network compression and layer-chaining techniques. The network compression technique is to make a style transfer CNN have low computational complexity and a small amount of parameters, and an efficient data compression method is proposed to reduce the feature map size. In addition, the layer-chaining technique is proposed to reduce the off-chip memory traffic and thus to increase the throughput at the cost of small hardware resources. In the proposed hardware architecture, a neural processing unit is designed by taking into account the proposed data compression and layer-chaining techniques. A prototype accelerator implemented on a FPGA board achieves a throughput comparable to the state-of-the-art accelerators developed for encoder-decoder CNNs.**

***Index Terms*— Convolutional neural network (CNN), style transfer application, compression, chaining, neural processing unit (NPU), field programmable gate array (FPGA).**

I. Introduction

CONVOLUTIONAL neural networks (CNNs) have widely been used in computer vision applications due to their superior accuracy [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. The computer vision applications can be classified into image-level labeling applications, such as image classification and object detection, and pixel-level labeling applications, such as super-resolution and image-to-image translation [25]. In the image-level labeling applications, an input image is processed by a feature extraction network to generate inferences for the objects in the image [1], [2], [3], [4], [5], [6]. In the pixel-level labeling applications, on the other hand, an input image is translated to another image of the same or higher resolution by using a encoder-decoder network that infers every pixel [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24].

Style transfer CNNs have been presented in [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], and [18] as a pixel-level labeling application. A style transfer CNN transforms the style of a content image into that of a style image, and it can be employed for digital picture frames or digital televisions to display artistic images generated from personal photographs. Recently, high-resolution images such as full high definition (FHD), quad high definition (QHD), and ultra high definition (UHD) images have been dominant in commercial products, so the implementation of a CNN processing high-resolution images is challenging, as the computational complexity and the feature map size are proportional to the input image size. Especially, the style transfer CNN is hard to be implemented as it employs an encoder-decoder network that maintains the feature map size. As the embedded processors employed in digital picture frames or digital televisions are not sufficient for the style transfer in general, an additional hardware platform is required to implement a style transfer CNN.

Hardware platforms such as general-purpose computing on graphics processing units (GPGPU), application-specific integrated circuit (ASIC), and field-programmable gate array (FPGA) are commonly used to implement CNNs. The GPGPU is a general-purpose platform mainly developed for computation-intensive applications, while the ASIC and FPGA are for specific applications. Although the GPGPU achieves high throughput due to massively parallel computing cores and high memory bandwidth, it consumes much more energy than the FPGA in processing vision applications [26]. There have been many works that implemented a CNN on an ASIC or FPGA to achieve high throughput with much less energy consumption [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53]. As the FPGA consisting of adaptive logic elements, digital signal processing (DSP) blocks, and block RAMs is configurable for specific hardware architecture and can provide high computing performance and

Manuscript received 13 September 2022; revised 27 November 2022 and 23 December 2022; accepted 3 January 2023. Date of publication 10 January 2023; date of current version 31 March 2023. This work was supported in part by the National Research Foundation of Korea under Grant NRF-2017R1E1A1A01076992; and in part by the Ministry of Science and ICT, South Korea, under Grant IITP-2020-0-01847. This article was recommended by Associate Editor M. Martina. *(Corresponding author: In-Cheol Park.)*

The authors are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea (e-mail: sckim@ics.kaist.ac.kr; icpark@kaist.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2023.3234640.

Digital Object Identifier 10.1109/TCSI.2023.3234640

energy efficiency, it has been applied even to commercial devices [54], [55]. There have been many techniques that enable CNNs to be implemented on the FPGA efficiently [56], [57], [58], [59], [60], [61], [62]. A lightweight network layer called a depthwise separable convolution was presented to lower the computational complexity and the parameter size [56]. A three-stage compression, which sequentially performs pruning, quantization, and compression, was developed to reduce the storage requirement [59].

There have been a few FPGA accelerators developed for style transfer applications [44], [45], [46]. For comprehensive comparison, other FPGA accelerators for super-resolution and denoising are also compared as their processing architectures are similar to the style transfer ones [39], [40], [41], [42], [43]. In [40], a super-resolution CNN that employs depthwise separable convolutions and 1-D kernels was developed to reduce the computational complexity. In addition, a quantization technique and a feature map compression technique were presented to implement hardware efficiently, while maintaining the peak signal-to-noise ratio (PSNR) performance. The quantization determines the fixed-point representations for weights and activations, and the compression is similar to S3 texture compression [63]. In [41], a super-resolution accelerator was proposed to increase the throughput by transforming deconvolutional layers into convolutional layers. In addition, a super-resolution CNN was compressed by quantizing 32-bit floating-point data into 13-bit fixed-point data, and by reducing the kernel size from 9×9 to 7×7. In [42], a super-resolution CNN that replaces a deconvolutional layer with an ESPCN layer [21] was proposed to reduce the number of parameters, and activations and weights were quantized to different fixed-point representations to maintain the PSNR performance. To balance the computation loads of pipeline stages, in addition, the input feature map and the kernel are separated into two parts, and the partial convolutions of the two separated parts are pipelined as they are independent of each other. Generative adversarial networks (GANs) have been employed for style transfer applications [44], [45], [46]. In [44] and [45], software frameworks were designed to generate optimized GAN accelerators with transposed convolutions. As a transpose convolution inserts a number of zeros into input feature maps, it induces many ineffectual operations. To eliminate those operations, the software framework reorganizes data by taking into account network configurations, resulting in higher performance and energy efficiency. In [46], a fast transformation algorithm was used to reduce the computational complexity, and a reconfigurable architecture was employed to support both convolution and deconvolution operations, which was verified by implementing a GAN on an FPGA.

Some previous works [39], [40], [41], [42], [43] have used the depth-first processing technique that processes the CNNs in the depth-first order instead of the layer-first order [47], [48], [49], [50], so as to reduce the off-chip memory access and increase the throughput. As the previous CNNs used are associated with a small number of layers and a small amount of parameters, the fully-pipelined architecture has been employed in realizing the depth-first processing technique. However, most CNNs developed for practical pixel-level labeling applications have many layers and a large amount of parameters [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [22], [23], [24], so it is hard to adopt the fully-pipelined architecture on an FPGA due to the lack of hardware resources. The fully-pipelined architecture needs a processing core for every layer, and should store all parameters and intermediate feature maps in the on-chip memory. Therefore, the depth-first processing technique cannot process all the layers in the depth-first order, and it is limited to only a couple of layers in realizing such CNNs on an FPGA. We call this approach a layer-chaining technique, where only several consecutive layers are tied in a chain and a layer in the chain generates an intermediate feature map and forwards it to the successive layer through a temporary on-chip memory. In the layer-chaining technique, the off-chip memory access decreases as the intermediate feature maps generated in the chained layers are not stored into the off-chip memory.

In this paper, we propose a CNN inference accelerator equipped with network compression and layer-chaining techniques to implement a style transfer CNN on an FPGA efficiently. The network compression technique modifies a style transfer CNN to reduce the computational complexity and the size of parameters, and a data compression method is proposed to reduce the feature map size. The layer-chaining technique is to reduce the off-chip memory traffic and improve the throughput with a small amount of on-chip memory. Employing the proposed techniques, a prototype CNN inference accelerator is designed and implemented on a FPGA board.

The rest of this paper is organized as follows. Section II reviews pixel-level labeling CNNs and previous FPGA implementations. The proposed network compression and layer-chaining techniques are explained in Section III and Section IV, respectively. Then, Section V describes the proposed hardware architecture. Section VI discusses and evaluates the FPGA implementation. Finally, concluding remarks are made in Section VII.

II. Background

A. Pixel-Level Labeling CNNs

There have been many CNNs designed for pixel-level labeling applications such as super-resolution and image-to-image translation including semantic segmentation, style transfer, and denoising. For super-resolution, SRCNN [19] was the first end-to-end CNN that upscales a low-resolution image into a high-resolution image directly. After that, FSRCNN [20] and ESPCN [21] were devised to reduce the complexity of SRCNN, while VDSR [22], SRResNet [23], and EDSR [24] were to improve the accuracy by increasing the number of layers. For image-to-image translation, a feedforward CNN was presented based on new perceptual loss functions that rely on high-level features [7], and StyleNet [8] employed instance normalization to improve the translation quality. To remove the retraining processes needed for new styles, universal style transfer CNNs, such as AdaIN [9], WCT [10], PhotoWCT [11], and WCT2 [12], took into account both content and style images. Several CNNs, such as pix-to-pix [15], CycleGAN [16], StarGAN [17], and DiscoGAN [18], addressed GANs [64] as a generic approach to learn the loss functions of various translation tasks.

Fig. 1 summarizes the number of multiply-accumulate (MAC) operations and that of parameters required for pixel-level labeling CNNs, where squares and circles represent

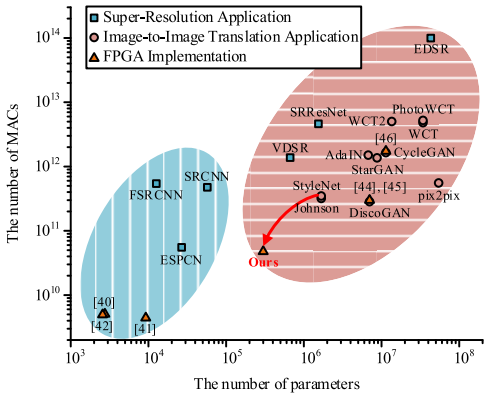

Fig. 1. Pixel-level labeling CNNs.

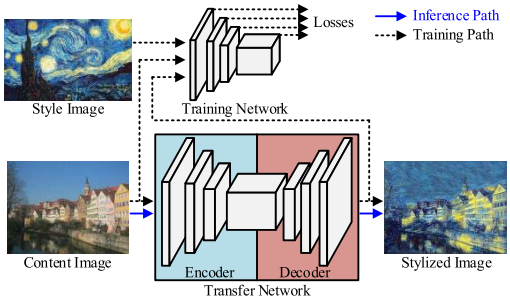

Fig. 2. Overall structure of a style transfer CNN.

super-resolution applications and image-to-image translation applications, respectively. According to the amount of parameters, the CNNs can be divided into two categories as denoted by patterned circles in Fig. 1. The right patterned circle includes large-scale CNNs in which the number of layers is larger than 10 and the number of parameters is larger than 10^5. On the other hand, the left patterned circle contains small-scale CNNs associated with smaller layers or parameters.

Fig. 2 shows a style transfer CNN called StyleNet, where solid lines and dotted lines represent inference paths and training paths, respectively. StyleNet is composed of two separate networks: one is a transfer network that transforms the style of a content image into the style of a style image, and the other is a training network that generates high-level features used to calculate losses in training. Only the transfer network is used in inference to convert a content image into a stylized image. The transfer network is a large-scale CNN consisting of 16 convolutional layers, and based on the encoder-decoder architecture where the encoder extracts features from a content image and the decoder reconstructs an output image having the pretrained style. In addition to style transfer, the transfer network has widely been used in pixel-level labeling applications such as super-resolution, object transfiguration, photo generation, photo enhancement, facial attribute transfer, and so on [7], [16], and [17]. Therefore, the transfer network is adopted as the baseline CNN for FPGA implementation in this work.

B. Previous FPGA Implementations

The previous FPGA implementations for super-resolution and denoising applications [39], [40], [41], [42], [43] have modified small-scale CNNs to reduce the computational complexity and the memory size required. For example, kernels are reduced in size, compute-intensive layers are replaced

TABLE I
CHARACTERISTICS OF FPGAs

Vendor	Xilinx	Xilinx	Xilinx	Xilinx
Device	Artix UltraScale+	Kintex UltraScale+	Virtex UltraScale+	Zynq UltraScale+ MPSoC
Logic Elements [K]	96-308	356-1843	862-8938	81-1143
DSP Blocks [Slices]	400-1200	1368-3528	1320-12288	216-3528
Block RAMs [MB]	0.4-1.3	1.6-7.6	3.0-11.8	0.5-4.3

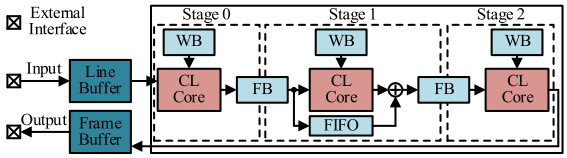

Fig. 3. Conventional hardware architecture for pixel-level labeling CNNs with the depth-first processing technique.

with lightweight layers, and feature maps are compressed. On the other hand, the previous FPGA implementations for style transfer applications [44], [45], [46] have used large-scale CNNs without any modification. Those implementations are denoted by triangles in Fig. 1. For FPGA implementation, the CNN is quantized to have a fixed-point representation so as to utilize the DSP blocks of the FPGA that are designed to support fixed-point arithmetic operators. The fixed-point representation denoted as (IL, FL) consists of an IL-bit integer and an FL-bit fraction, and its word length, WL, is the sum of IL and FL.

Fig. 3 exemplifies a conventional hardware architecture with the depth-first processing technique, which consists of three convolutional layers and a residual connection, where CL, WB, and FB stand for a convolutional layer, a weight buffer, and a feature map buffer, respectively. The line buffer forwards the pixel values of an input image to the first CL core, and the frame buffer stores an output image from the last CL core. Each CL core processes a convolutional layer, and three CL cores are connected in a pipelined manner so that a core feeds the next core, constructing the 3-stage fully-pipelined architecture. Three types of on-chip memory, WB, FB, and FIFO, are used: the WB stores parameters, the FB temporarily saves the feature map to be delivered to the next core, and the FIFO supports the residual connection by storing the feature map to be accumulated with another feature map. The WB stores all the parameters of the three layers to process them simultaneously, while the FB and the FIFO store a part of the feature map, as the feature map is processed as soon as it is generated.

The fully-pipelined architecture was designed for the depth-first processing technique. However, it is hard to apply the fully-pipelined architecture to large-scale CNNs because of the limited hardware resources available in an FPGA. The size of parameters is more than several MBs, and the feature maps to be stored is as large as the receptive outputs that are proportional to the number of layers. In addition, the sizes of the line and frame buffers should be large enough to deal with high-resolution images. To store an FHD image, a frame buffer needs an on-chip memory of about 6MB. Table I summarizes the characteristics of Xilinx UltraScale+ FPGAs [65]. The block RAM size available on an FPGA is not so large that the

TABLE II
LAYER DETAILS OF STYLENET

Layer	w_i	h_i	c_i	c_o	k	s $(1/u)$
CL	1920	1080	3	32	9	1
CL	1920	1080	32	64	3	2
CL	960	540	64	128	3	2
RB	480	270	128	128	3	1
RB	480	270	128	128	3	1
RB	480	270	128	128	3	1
RB	480	270	128	128	3	1
RB	480	270	128	128	3	1
DL	960	540	128	64	3	1/2
DL	1920	1080	64	32	3	1/2
CL	1920	1080	32	3	9	1

TABLE III
LAYER DETAILS OF THE PROPOSED CNN

Layer	w_i	h_i	c_i	c_o	k	s $(1/u)$
CL	1920	1080	3	32	3	1
DSCL	1920	1080	32	64	3	2
DSCL	960	540	64	128	3	2
DSRB	480	270	128	128	3	1
DSRB	480	270	128	128	3	1
DSRB	480	270	128	128	3	1
DSRB	480	270	128	128	3	1
DSRB	480	270	128	128	3	1
DSDL	960	540	128	64	3	1/2
DSDL	1920	1080	64	32	3	1/2
CL	1920	1080	32	3	3	1

TABLE IV
NOTATIONS SPECIFYING A CNN

Notation	Description
w	Input width of a layer
h	Input height of a layer
c_i	Input channel number of a layer
c_o	Output channel number of a layer
k	Kernel width and height of a layer
s	Stride size of a layer
p	Padding size of a layer
u	Upscale size of interpolation

conventional hardware architecture is not suitable for realizing large-scale CNNs even on the modern FPGAs.

III. NETWORK COMPRESSION TECHNIQUE

This section first describes the CNN modified to reduce the computational complexity and the number of parameters. Then, a new data compression method devised to reduce the feature map size is explained.

A. Network Architecture

Tables II and III summarize network architectures of StyleNet [8] and the proposed CNN, respectively, where CL, RB, DL, DSCL, DSRB, and DSDL denote a convolutional layer, a residual block, a deconvolutional layer, a depthwise separable convolutional layer, a depthwise separable residual block, and a depthwise separable deconvolutional layer. The details of the blocks used in the proposed CNN are illustrated in Fig. 4, and the notations are summarized in Table IV. The proposed CNN consists of 30 convolutional layers, and uses the instance normalization, the rectified linear unit (ReLU) activation function, and the nearest-neighbor interpolation in which an interpolated position has the same value as its nearest neighbor. As shown in Table III, the proposed CNN has a symmetric encoder-decoder network in which the front two DSCLs reduce the feature map size with a stride of 2, the five RBs maintain the feature map size, and the following two

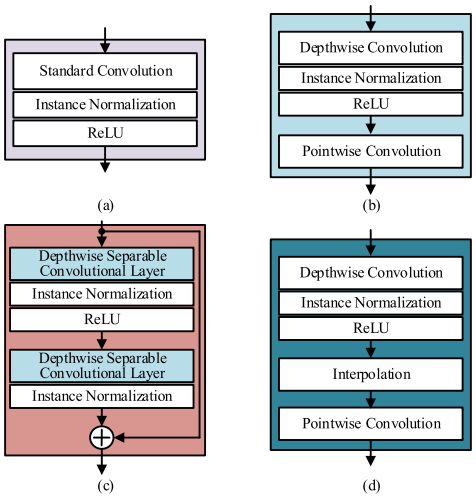

Fig. 4. Details of the blocks used in the proposed CNN. (a) A convolutional layer, (b) a depthwise separable convolutional layer, (c) a depthwise separable residual block, and (d) a depthwise separable deconvolutional layer.

DSDLs extend the feature map size by performing the nearest-neighbor interpolation with a scaling factor of 2.

Compared to StyleNet, the proposed CNN replaces standard convolutional layers with depthwise separable convolutional layers except for the first and last layers being maintained not to degrade the accuracy. In computer vision applications, depthwise separable convolutional layers lead to little accuracy drop [56]. As discussed in Section VI, the proposed CNN also maintains the accuracy while reducing the computational complexity and the size of parameters compared to SyteNet. However, replacing a standard convolution with a depthwise separable convolution increases the total size of feature maps generated. The standard convolution outputs a single feature map, while the depthwise separable convolution generates two feature maps, one from a depthwise convolution and the other from a pointwise convolution. The more feature maps may lead to the larger memory access. To resolve the problem, the layer-chaining technique is proposed in Section IV. In addition, the kernel sizes of the first and last convolutional layers are reduced from 9 to 3 as the number of hidden layers is large enough to reflect the entire input image with the reduced kernel size. The receptive field obtained by activating the 5th residual block is 6561×6561 due to the 8 preceding layers with a kernel size of 3. The proposed CNN is also denoted in Fig. 1.

B. Data Compression

Fig. 5 shows some characteristics of the feature maps generated by the proposed CNN. Fig. 5(a) shows the sparsity of the feature maps generated by the proposed CNN. The previous CNN accelerators have employed sparsity-based data compression methods such as run-length encoding. In the proposed CNN, however, the sparsity of the feature maps is low as the ReLU activation function is applied partially. The average sparsity is 31%, which means that the feature maps are usually non-zero, making sparsity-based compression and calculation ineffective for the proposed CNN. Fig. 5(b) shows the histograms of the input, the intermediate feature map, and the output. The input and the output features range from 0 to 255, whereas the intermediate features are usually very small where most are less than 8 due to small-valued kernels.

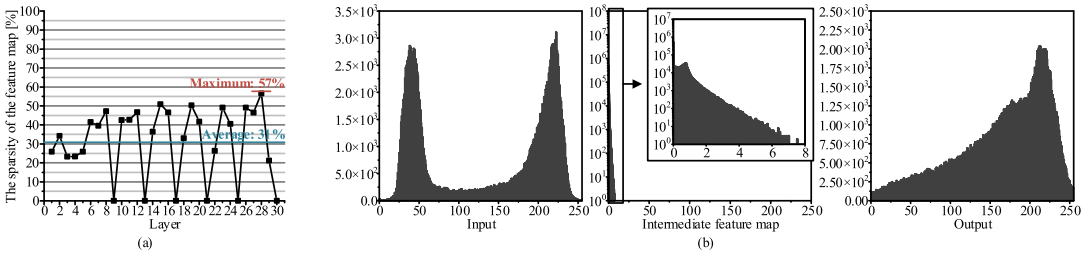

Fig. 5. Characteristics of the proposed CNN. (a) The sparsity of the feature map, and (b) the distributions of data.

In the proposed data compression method, the activations and weights represented in two's complement number system are quantized to predefined fixed-point representations. The activations and the weights have different ranges, so they are expressed with different fixed-point representations. The activation requires a large IL to express image data, while the weight needs a large FL as the trained weight is usually less than 1. After the activation is multiplied by the weight, the product size is extended to the sum of activation WL and weight WL, needing to be quantized to the same fixed-point representation as the activation. When converting to the fixed-point representation, the activation and the weight can have different rounding schemes. The activation is truncated as the rounding to the nearest value needs additional hardware resources, while the weight is rounded to the nearest value as the weight rounding can be done in advance. If there is an overflow or underflow, both the activation and the weight are clipped to the maximum or minimum representable value in order to maintain the accuracy.

Fig. 6 shows an example of the proposed data compression method, where q_{ij}, c_{ij}, and d_{ij} are a quantized activation, a compressed activation, and a decompressed activation, respectively. The quantized feature map is partitioned into feature blocks of the same size. A feature block in Fig. 6 is a group of 2×2 activations, and each block is compressed independently of the other blocks. In a feature block, we first search for the effective most significant bit (EMSB), and keep the significant length (SL) bits from the EMSB while truncating the rest bits for compression. The SL size is another parameter denoting the number of bits to be preserved. In Fig. 6, the EMSB is bit 6 and the SL is 4, so 4 bits from bit 6 to bit 3 are preserved. For decompression, the shift amount representing the number of truncated bits is stored along with the compressed activations. In addition, the block type is stored for further compression.

Fig 7 shows three possible block types and how to determine the EMSB, where block type 0 is used when the signs of activations are mixed, and block types 1 and 2 are used when the activations are all positive and all negative, respectively. For block type 0, the EMSB is the highest bit among least sign bits of the activations in a group as shown in Fig. 7(a), where the least sign bits are denoted by the dotted boxes. For block types 1 and 2, on the other hand, the EMSB is one bit lower than that for block types 0 as shown in Figs. 7(b) and 7(c) because the signs can be easily restored by referring to the block type in decompression.

In the decompression process shown in Fig. 6, the compressed activations are logically shifted left by the shift amount, which is in fact realized by padding zeros to the lower truncated bits. The upper truncated bits are restored by extending the sign. The sign bit of every activation is extended for block type 0. The sign bit is determined to 0 for block type 1 as the activations are all positive, and 1 for block type 2 as they are all negative.

The proposed data compression method truncates insignificant bits, so the compression noise is quite low as the significant bits in the block are preserved. In Fig. 5(b), for example, the upper bits are removed in the intermediate feature maps as the activations have small values, while they are preserved in the input and the output. The two configurable parameters, the size of a feature block and the SL size, make a tradeoff between the compression ratio and the accuracy. The compression ratio of the proposed data compression method is calculated as

$$\begin{aligned}&\text{Compression Ratio}\\&= \frac{\text{Uncompressed Data}}{\text{Compressed Data}}\\&= \frac{\text{Feature Block Size} \times \text{WL}}{\text{Feature Block Size} \times \text{SL} + \text{Block Type} + \text{Shift Amount}},\end{aligned} \tag{1}$$

where the shift amount is $\log_2(\text{WL})$ bits, and the block type is two bits. In Fig. 6, the block size is 4, the WL is 8 bits, the SL size is 4 bits, the shift amount is 3 bits, and the block type is 2 bits, so the compression ratio is about 1.5. The compression ratio can be increased by increasing the block size or decreasing the SL size.

A data compression method for the super-resolution CNN was presented in [40]. The method finds the maximum value among the activations in a block, and calculates the interpolated values between the maximum value and zero so as to build a table. Then, an activation in the block is replaced with the table index associated with the closest value to the activation. In the approach, a compressed block consists of the indexes replacing its activations and the maximum activation value of the block. The proposed method requires computational complexity of $O(n)$ to determine the block type, where n is the number of activations in the feature block, $O(n)$ to find the EMSB index, and $O(n)$ complexity to truncate the activations. On the other hand, the previous method needs $O(n)$ complexity to obtain the maximum value, $O(m)$ complexity to build the table, where m is the number of entries in the table, and $O(n \times m)$ complexity to replace the activations in the block with the indexes. The proposed compression needs less computational complexity in compression and decompression than the previous method.

IV. Layer-Chaining Technique

In the layer-chaining technique, how layers are chained influences the off-chip memory access and the on-chip memory size as the feature maps generated in layers are different

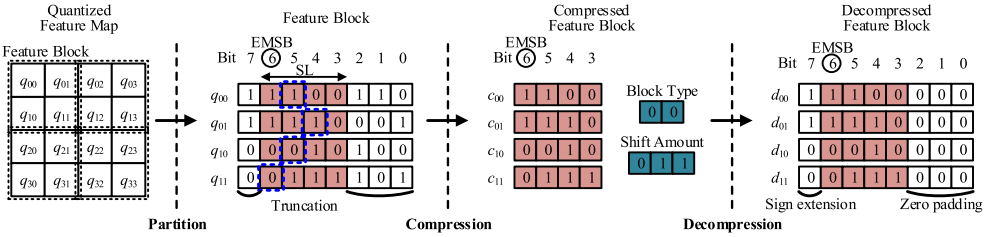

Fig. 6. Example of the proposed data compression and decompression.

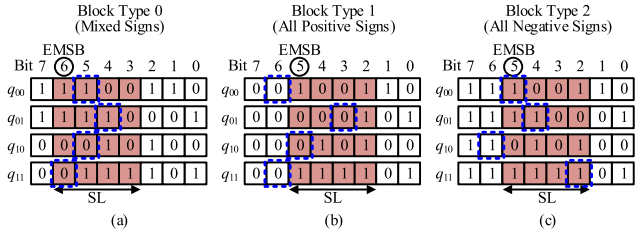

Fig. 7. Types of feature blocks. (a) A mixed-sign block, (b) an all-positive block, and (c) an all-negative block.

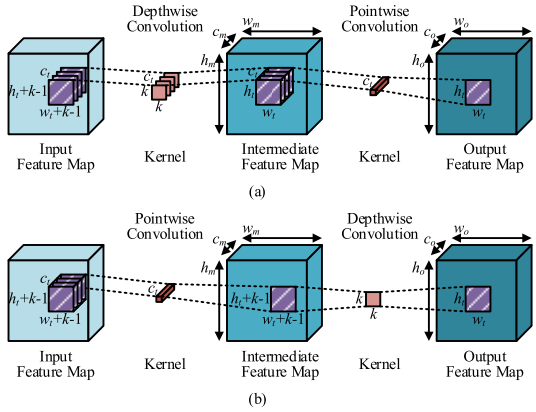

Fig. 8. Two layer-chaining methods. (a) Depthwise-and-pointwise chaining, and (b) pointwise-and-depthwise chaining.

in size. For efficient hardware realizations, therefore, it is required to analyze the combinations of layers to be chained. In this section, two layer-chaining methods are analyzed in terms of on-chip memory size, off-chip memory access, and processing time. As the on-chip memory supported in an FPGA is not large enough to store the entire feature map, a feature map is assumed to be processed using the tiling technique that divides the feature map into several tiles and processes each tile separately.

A. Layer-Chaining Methods

The layer-chaining technique ties a couple of successive layers in a chain to process the layers without accessing the off-chip memory. As a dedicated hardware is designed to process such a chain, the chain should be built such that it can be highly reused in processing a CNN. Although increasing the number of chained layers leads to less off-chip memory accesses, it needs more hardware resources. If the number of chained layers increases, the size of a tile should decrease as the hardware resources available on an FPGA is restricted, increasing the redundant computations caused by the overlapped data region among tiles. Considering the computational overhead and the hardware resources available on an FPGA, the number of layers to be chained is limited to two in this work.

The proposed CNN is mainly composed of depthwise separable convolutional layers, each of which has a depthwise convolutional layer and a pointwise one. Fig. 8 exemplifies two layer-chaining methods where both convolutions have a stride size of one, patterned boxes represent feature tiles to be processed, and subscripts m, o, and t stand for intermediate, output, and tile, respectively. The other notations are summarized in Table IV. Fig. 8(a) shows the conventional depthwise-and-pointwise chaining method, where a depthwise convolution is followed by a pointwise convolution [51]. The depthwise convolution produces a set of 2-D intermediate feature tiles by using a set of 2-D kernels and a set of 2-D input feature tiles, and the pointwise convolution completes a 2-D output feature tile by using a pointwise kernel and the set of 2-D intermediate feature tiles generated by the previous depthwise convolution. Fig. 8(b) illustrates another chaining method, pointwise-and-depthwise chaining, where a pointwise convolution is followed by a depthwise convolution. The pointwise convolution processes a set of 2-D input feature tiles to compute a 2-D intermediate feature tile, and the depthwise convolution applies a 2-D kernel to the 2-D intermediated feature tile to calculate a 2-D output feature tile.

B. On-Chip Memory Size

In the layer-chaining technique, the intermediate feature tile generated by the front process should be stored into the on-chip memory as it will be processed immediately by the rear process. The depthwise and the pointwise convolutional layers are different in their input shapes, so the two layer-chaining methods are associated with different on-chip memory sizes. In Fig. 8(a), the sizes of the input feature tile, the intermediate feature tile, and the output feature tile are $(w_t+k-1)\times(h_t+k-1)\times c_t$, $w_t\times h_t\times c_t$, and $w_t\times h_t$, respectively. The total tile size is approximately $w_t\times h_t\times(2c_t+1)$, and the kernel size is $k\times k\times c_t+1\times 1\times c_t$. In Fig. 8(b), on the other hand, those sizes are $(w_t+k-1)\times(h_t+k-1)\times c_t$, $(w_t+k-1)\times(h_t+k-1)$, and $w_t\times h_t$, so the total size is roughly $w_t\times h_t\times(c_t+2)$, and the kernel size is $1\times 1\times c_t+k\times k$.

The characteristics of two chaining methods are summarized in Table V, where (w_t+k-1) and (h_t+k-1) are approximated to w_t and h_t for simplicity. The input feature size and the output feature size are the same for the two chaining methods, but the intermediate feature size is c_t times larger in the depthwise-and-pointwise chaining method. In Fig. 8(a), the set of 2-D intermediate feature tiles is stored into the on-chip memory as the following pointwise convolution requires all the channels. In Fig. 8(b), on the other hand, only the 2-D intermediate feature tile is stored into the on-chip memory as the depthwise convolution needs only a channel at a time. The total feature size required to be stored into the on-chip memory is about 2 times larger in the depthwise-and-pointwise chain-

TABLE V
CHARACTERISTICS OF DEPTHWISE-AND-POINTWISE (DaP) AND POINTWISE-AND-DEPTHWISE (PaD) CHAINING METHODS

Method	DaP Chaining	PaD Chaining
Input Feature Map Size	$w_t \times h_t \times c_t$	$w_t \times h_t \times c_t$
Intermediate Feature Map Size	$w_t \times h_t \times c_t$	$w_t \times h_t$
Output Feature Map Size	$w_t \times h_t$	$w_t \times h_t$
Total Feature Map Size	$w_t \times h_t \times (2c_t + 1)$	$w_t \times h_t \times (c_t + 2)$
Total Kernel Size	$k \times k \times c_t + c_t$	$c_t + k \times k$
MACs in Depthwise Convolution	$k \times k \times w_t \times h_t \times c_t$	$k \times k \times w_t \times h_t$
MACs in Pointwise Convolution	$1 \times 1 \times c_t \times w_t \times h_t$	$1 \times 1 \times c_t \times w_t \times h_t$

ing method. In the pipelined architecture, the on-chip memory for the feature tiles should be designed based on the double-buffering technique in order to make the front process store new results while the rear process load the previous results. In addition, the total kernel size is also larger in the depthwise-and-pointwise chaining method as shown in Table V.

C. Off-Chip Memory Access

In the layer-chaining technique, the output feature tiles of a chain are stored into the off-chip memory, and are loaded from the off-chip memory to start the processing of the next chain. In Fig. 8(a), the off-chip memory is accessed to write and read the feature tiles generated by the pointwise convolution, and the results of the depthwise convolution is stored in the on-chip memory. In Fig. 8(b), on the other hand, the results of the depthwise convolution are stored into the off-chip memory. For the analysis of off-chip memory accesses, all the required computations for a feature map are assumed to be performed when the feature map is read. Even if multiple off-chip memory accesses are required in some cases, the amount of off-chip memory accesses is proportional to the feature size, so the analysis still works.

In Fig. 8(a), the output feature map size $w_o \times h_o \times c_o$ is usually larger than the intermediate feature map size $w_m \times h_m \times c_m$. One reason is that the pointwise convolution expands the number of channels, i.e., $c_m < c_o$ in the encoder, and the second reason is that the spatial size of the feature map increases, i.e., $w_m \times h_m < w_o \times h_o$ due to the interpolation performed right before the pointwise convolution in the decoder. In the depthwise-and-pointwise chaining method, the size of feature maps to be stored in the off-chip memory is accordingly larger than the size of the intermediate feature map to be stored in the on-chip memory. In Fig. 8(b), on the other hand, the output feature map size $w_o \times h_o \times c_o$ is smaller than the intermediate feature map size $w_m \times h_m \times c_m$. This is because the depthwise convolution maintains the same number of channels, i.e., $c_m = c_o$, and the spatial size of an intermediate feature map is reduced due to the stride, i.e., $w_m \times h_m > w_o \times h_o$. In the pointwise-and-depthwise chaining method, therefore, the size of feature maps to be stored in the off-chip memory is smaller. As a result, the amount of off-chip memory accesses is larger in the depthwise-and-pointwise chaining method than that in the pointwise-and-depthwise chaining method.

D. Processing Time

In processing a depthwise separable convolutional layer, the heterogeneous architecture that has separate processing units for depthwise convolution and pointwise convolution outperforms the homogeneous architecture with a unified processing unit in terms of throughput and off-chip memory traffic [51]. In the depthwise separable convolutional layer, the pointwise convolutional layer has a larger number of MACs than the depthwise convolutional layer has. Fig. 9 shows the number of MACs required in each layer of the proposed CNN to process an FHD image. In the heterogeneous architecture, therefore, more MAC operators are assigned to the pointwise convolution processing block [52].

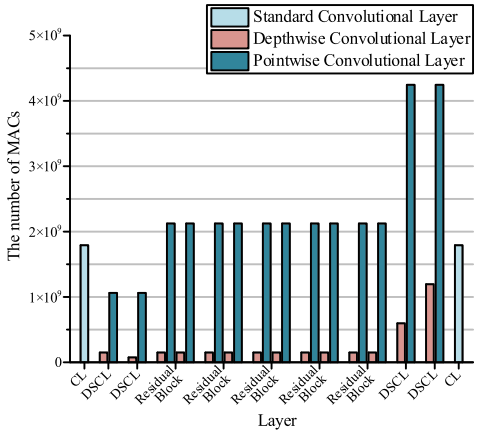

Fig. 9. Number of MACs required in each layer of the proposed CNN to process an FHD image.

In Fig. 8(a), the number of MACs in the depthwise convolution is larger than that in the pointwise convolution. In other words, the depthwise convolution requires $k \times k \times w_t \times h_t \times c_t$ MACs, while the pointwise convolution needs $1 \times 1 \times c_t \times w_t \times h_t$ MACs. In the depthwise-and-pointwise chaining method, the two processing blocks are unbalanced in processing time as the number of MACs in the pointwise processing is much smaller than that in the depthwise processing. In Fig. 8(b), on the other hand, the pointwise convolution requires the same number of MACs as in Fig. 8(a), while the depthwise convolution needs a smaller number of MACs, $k \times k \times w_t \times h_t$. Therefore, the two processing blocks are more balanced in the pointwise-and-depthwise chaining method. The characteristics of the two chaining methods are summarized in Table V.

V. HARDWARE ARCHITECTURE

The analysis in Section IV shows that the pointwise-and-depthwise chaining method is superior to the conventional depthwise-and-pointwise chaining method in terms of the size of on-chip memory, the number of off-chip memory accesses, and the overall processing time. Therefore, the pointwise-and-depthwise chaining method is employed in the proposed hardware architecture. In this section, we describe the hardware architecture developed to accelerate the proposed CNN. The overall hardware architecture is first addressed, and then the neural processing unit (NPU) that realizes the proposed data compression and chaining methods is explained.

A. Overall Architecture

Fig. 10 shows the overall block diagram of the proposed hardware architecture, where dotted and solid lines represent control channels and data channels, respectively. The host processor, which is a CPU with cache memories, manages

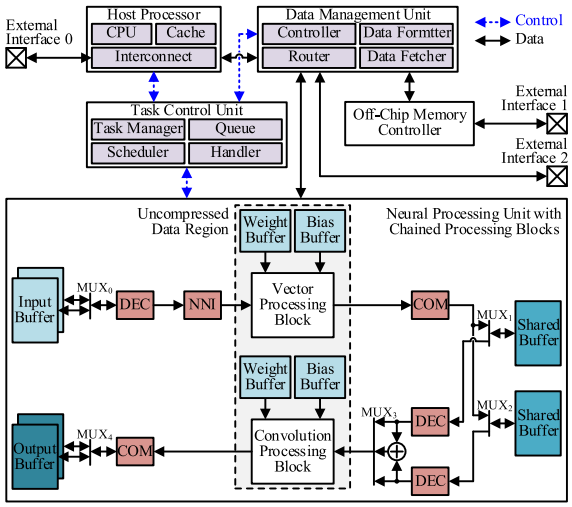

Fig. 10. Overall block diagram of the proposed hardware architecture.

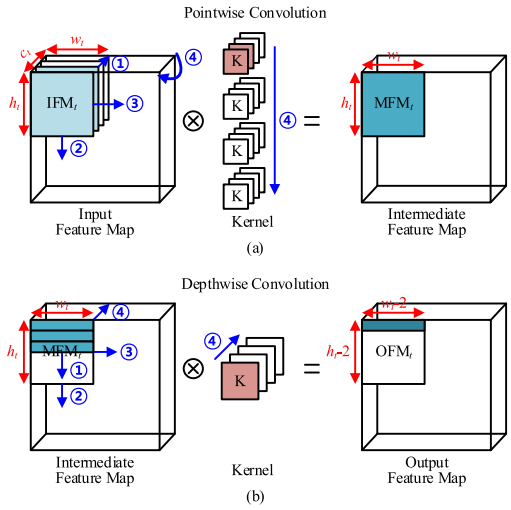

Fig. 11. Computations and processing flows of two processing blocks. (a) A vector processing block and (b) a convolution processing block.

the task control unit composed of a task manager, a command queue, a scheduler, and an interrupt handler. The task control unit supervises the data management unit and the NPU through the control channel. The data management unit consisting of a data fetcher, a data formatter, a router, and a controller performs data formatting and data transfer. It is connected to the host processor, the NPU, and the off-chip memory controller through dedicated data channels. Details on the task control unit and the data management unit can be found in [52]. In addition, there are three external interfaces that are separately linked with the host processor, the off-chip memory controller, and the data management unit.

B. NPU Architecture

In the NPU architecture shown in Fig. 10, COM, DEC, and NNI denote a compressor, a decompressor, and a nearest-neighbor interpolator. In addition to those blocks, the NPU has two processing blocks and several on-chip memories. The vector processing block is tailored to pointwise convolution, while the convolution processing block is developed for 3-D convolution and depthwise convolution. Their details are elaborated in [52]. However, the two blocks are chained to run simultaneously in a pipelined manner, realizing the pointwise-and-depthwise chaining method. Each processing block has its own weight and bias buffers that supply parameters, and the feature tiles are stored in input, shared, and output buffers. The compressor and the decompressor conduct data compression and decompression so as to reduce the on-chip memory size and the off-chip memory traffic. For deconvolution, the nearest-neighbor interpolator upscales the feature tile by assigning an interpolated activation the value of the nearest horizontal neighbor at the same vertical address of the memory.

The input buffer provides the vector processing block with the input feature tile, the shared buffer is to transfer the intermediate feature tile between the two processing blocks, and the output buffer stores the output feature tile resulting from the convolution processing block. The buffers are connected to the data management unit through the data channel. The input buffer is double-buffered to hide the loading latency from the off-chip memory, and the MUX_0 selects one of the buffers to be processed. The shared buffer is also double-buffered, so one is connected to the vector processing block while the other to the convolution processing block, enabling the two processing blocks to operate in a pipelined manner. The shared buffers are controlled by MUX_1, MUX_2, and MUX_3. MUX_1 and MUX_2 select the buffer to be connected to either the vector processing block or the convolution processing block, and MUX_3 selects the input of the convolution processing block, where the second input denoting the sum of the shared buffers is used to realize a residual connection. The output buffer is double-buffered as well, and controlled by MUX_4.

The system depicted in Fig. 10 compresses data except for the uncompressed data region including the vector and the convolution processing blocks and the weight and bias buffers. The uncompressed region includes the processing blocks to maintain the accuracy by computing the feature tile in a high precision, while it excludes the off-chip memory and the input, shared, output buffers to reduce off-chip memory accesses and the on-chip memory size required. The compressors and the decompressors wrap the uncompressed region. The weight and bias buffers can be compressed to further decrease their sizes. As the weight and bias buffers are much smaller than the buffers for the feature maps, however, it has a little effect on the overall hardware complexity. Compressing only the feature tile is effective in reducing the on-chip memory size as well as the off-chip memory traffic, and makes a good tradeoff in reducing adaptive logic elements required.

Fig. 11 shows the computations performed in the processing blocks and the processing flow for a pointwise-and-depthwise chain, where IFM_t, MFM_t, OFM_t, K, w_t, h_t, and c_t are an input feature tile, an intermediate feature tile, an output feature tile, a kernel, tile width, tile height, and the number of channels, respectively. In processing a tile, boxes with the same color are computed in parallel. The vector processing block processes several channels of an IFM_t in parallel, and the convolution processing block computes several lines of a MFM_t simultaneously. For multiple cycles, the vector processing block generates a $w_t \times h_t$ MFM_t by associating a $w_t \times h_t \times c_t$ IFM_t with a $1 \times 1 \times c_t$ kernel, and the convolution processes block processes the $w_t \times h_t$ MFM_t, generated by the vector processing block, with a 3×3 kernel, resulting a $(w_t - 2) \times (h_t - 2)$ OFM_t. Note that the processing

TABLE VI
CHARACTERISTICS OF THE PROPOSED FPGA IMPLEMENTATION

Hardware Platform	Terasic Han Pilot Platform
FPGA	Intel Arria 10 SoC 10AS066K3F40E2SG
Technology	20nm
Maximum Clock Frequency	200MHz
ALMs (Utilization)	189K (75%)
DSP Blocks[a] (Utilization)	1536 (91%)
Block RAMs (Utilization)	3.72MB (68%)

[a]A DSP block is configured to have two 18×19 multipliers.

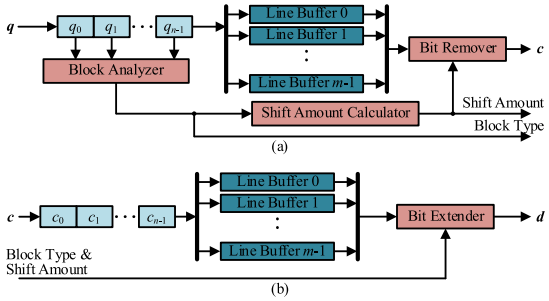

Fig. 12. Hardware architecture for the proposed data compression. (a) A compressor and (b) a decompressor.

blocks are configured such that the output shape of the vector processing block is the same as the input shape of the convolution processing block to minimize redundant computations. This processing flow within a tile is denoted by the circled number 1 in Fig. 11.

Tiles in the vertical direction are processed first, which is denoted by the circled number 2, and then neighbor tiles in the horizontal direction are processed as denoted by the circled number 3. The same flow is repeated until the entire input feature map is processed in the pointwise convolution, which corresponds to a channel of the intermediate feature map in the depthwise convolution. Afterward, the processing flow described above is repeated for other kernels in the pointwise convolution and other channels in the depthwise convolution, which is denoted by the circled number 4.

The overall structures of the proposed compressor and decompressor are illustrated in Fig. 12. The input is a 1-D vector of length n, and m line buffers of length n are used to compress and decompress a $m \times n$ feature block. In the compressor shown in Fig. 12(a), the block analyzer finds the block type and the EMSB in a $m \times n$ feature block while the feature block is stored in the m line buffers for m cycles. Then the shift amount calculator computes the shift amount by counting the number of bits to be truncated, which is used in the bit remover. In the decompressor shown in Fig. 12(b), the m line buffers store the compressed feature block, and decompression is simply achieved by padding zeros and extending the sign.

VI. IMPLEMENTATION AND EVALUATION

In this section, we first describe a prototype accelerator implemented on a FPGA board. Then, it is evaluated and compared with previous FPGA implementations.

A. FPGA Implementation

The proposed hardware architecture designed in Verilog HDL was implemented on a Terasic HAN Pilot Platform embedding an Intel Arria 10 SoC FPGA, where an adaptive logic element is called an adaptive logic module (ALM). Table VI summarizes the characteristics of the FPGA implementation. The maximum operating frequency is 200MHz, 189K ALMs are used to realize combinational logics, registers, and buffers, 1536 DSP blocks are utilized for fixed-point arithmetic operators, and about 3.7MB block RAMs are employed for on-chip memory. The utilizations of ALMs, DSP blocks, and block RAMs are about 75%, 91%, and 68%, respectively. Fig. 13 shows the prototype accelerator developed for style transfer applications. The FPGA board in Fig. 13(a) denotes the components used in the prototype accelerator. A demonstration is shown in Fig. 13(b), where the prototype accelerator transfers an image in the SD card into another style and displays the style-transferred image on the monitor thorough the HDMI interface. Fig. 13(c) represents the overall block diagram of the prototype accelerator, where the dotted box on the right indicates the board components shown in Fig. 13(a), and the solid boxes on the left denote the hardware blocks implemented on the Arria 10 SoC FPGA.

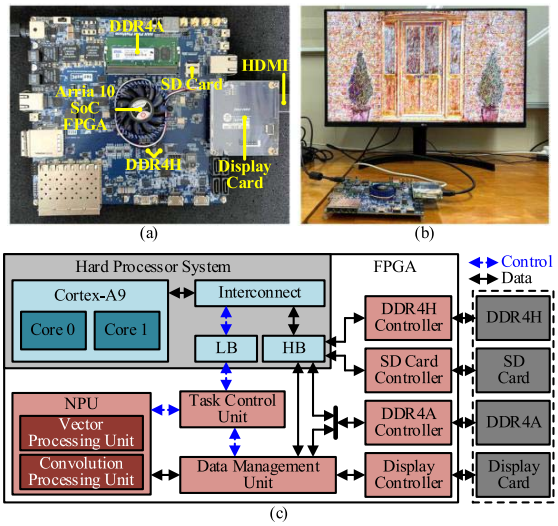

Fig. 13. Prototype style-transfer accelerator. (a) The FPGA board, (b) a snapshot of a demonstration, and (c) an overall block diagram.

As shown in Fig. 13(c), the Arria 10 SoC FPGA is divided into two parts: the first is the hard processor system implemented as a hard IP and the second is the FPGA reconfigurable for a custom design. The hard processor system containing a dual-core ARM Cortex-A9 processor is used as the host processor. The Cortex-A9 communicates with the FPGA part through the low-bandwidth (LB) interface and the high-bandwidth (HB) interface. The low-bandwidth interface works as a control channel for the task control unit. The high-bandwidth interface is used to transfer data and linked with the DDR4H memory, the DDR4A memory, the SD card, and the data management unit. The DDR4H is mainly used for the main memory of the Cortex-A9, and the DDR4A is to store parameters and feature maps required for the NPU. The SD card contains an FHD image to be style-transferred and the parameters to be loaded to the DDR4A when booting the prototype accelerator. The display controller connected to the data management unit displays the inference results generated by the NPU.

In the NPU, the vector processing block computes an input feature tile of $64 \times 18 \times 2$ in a cycle, and the convolution processing block calculates an intermediate feature tile of

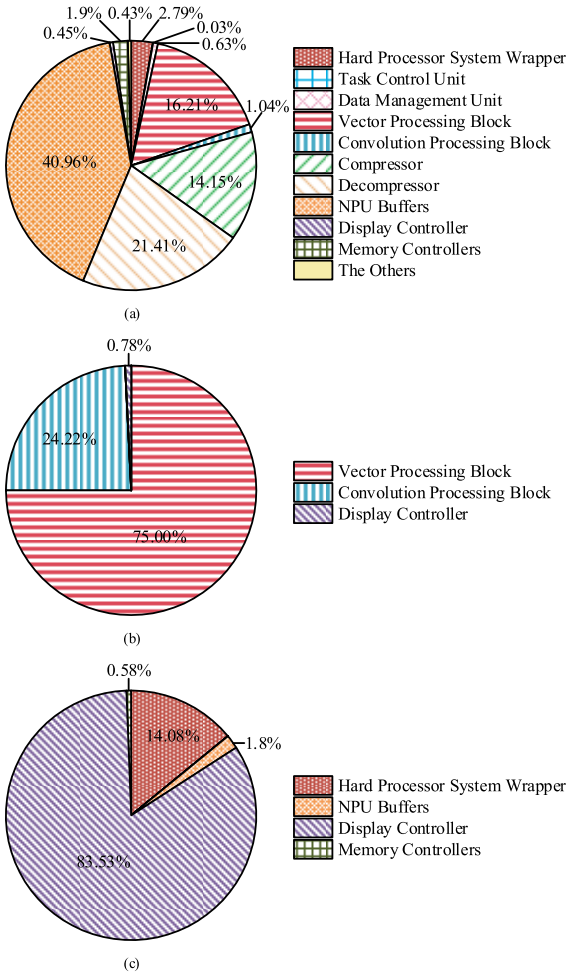

Fig. 14. Details of the proposed FPGA implementation. Breakdowns of (a) ALMs, (b) DSP blocks, and (c) block RAMs.

64 × 3 in a cycle. The sizes of the input buffer, the shared buffer, and the output buffer are configured as 186KB, 2.3KB, and 2.0KB, respectively, in order to make them large enough to store the data required in processing a tile. The sizes of the weight buffer and the bias buffer in the vector processing block are 66KB and 256B respectively, and these in the convolution processing block are 2.3KB and 256B. Fig. 14 shows details of the FPGA implementation. In Fig. 14(a), about 53% of the ALMs is used for the combinational logics in the NPU including the vector processing block, the convolution processing block, the compressors, and the decompressors, and about 41% for the input, the shared, and the output buffers in the NPU. In Fig. 14(b), most DSP blocks are devoted to the vector processing block and the convolution processing block, accounting for about 75% and 25%, respectively. In Fig. 14(c), most block RAMs are used to realize the frame buffer of the display controller. The frame buffer is large enough to store an FHD image to be displayed on the monitor. Some block RAMs are used to make asynchronous FIFOs that works as communication channels between the hard processor system and the FPGA. Block RAMs are also consumed to realize the weight and bias buffers in the NPU.

B. Evaluation

The proposed CNN replaces standard convolutional layers with depthwise separable convolutional layers and adjusts the

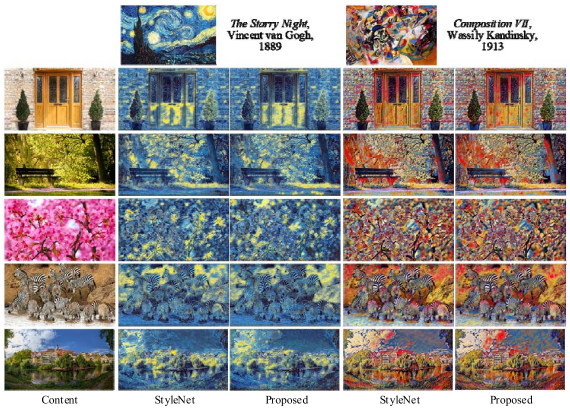

Fig. 15. Visual evaluation for StyleNet and the proposed CNN.

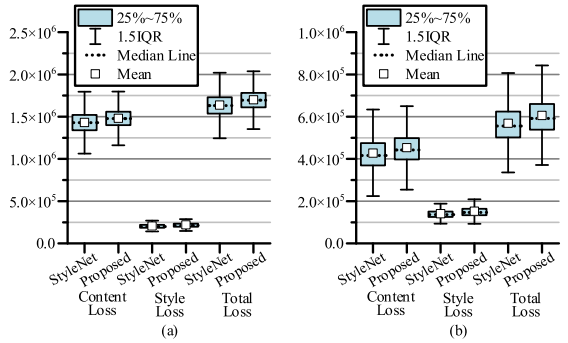

Fig. 16. Losses in processing the COCO dataset for the styles of (a) The Starry Night, and (b) Composition VII.

kernel size in the first and the last layers, reducing the number of MAC operations and the total parameter size to 12.2% and 12.1% of StyleNet, respectively. Note that the training method employed for the proposed CNN is exactly the same as that for StyleNet. For a visual evaluation, the stylized images generated by StyleNet and the proposed CNN are shown in Fig. 15. The quality of the style-transferred images resulting from the proposed CNN is almost identical to that of StyleNet. For a quantitative evaluation, the box-and-whisker plots of the losses calculated from the training network in processing the COCO dataset [66] are depicted in Fig. 16, where content loss indicates the structure similarity between a content image and a stylized image, style loss represents the style similarity between a style image and a stylized image, and total loss is the sum of them. The proposed CNN increases the total losses by about 5% on average for two styles, but the increase seems to be insignificant in style transfer applications.

For quantization, the WL of the fixed-point representation is decided by considering the accuracy and the precision of the arithmetic operators in the DSP block. The DSP block is dedicated for high-precision fixed-point arithmetic operators [67], [68]. The WL should be large enough to minimize quantization errors. However, a high precision leads to more off-chip memory traffic and hardware resources. In addition, a DSP block of the intel FPGA can have two multipliers if the size is not greater than 18 × 19 bits. Therefore, the WL is determined to be 16 bits by considering the ranges of activation and weight values, which is sufficient to maintain the accuracy. Applying the proposed quantization scheme, the IL and the FL of the activation representation are set to 9 and 7 bits, and those of the weight representation to 5 and 11 bits.

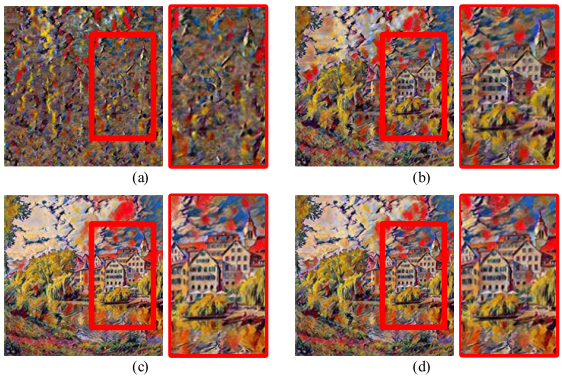

Fig. 17. Visual evaluation of the proposed CNN for the Composition VII style when the significant length is set to (a) 2 bits, (b) 3 bits, (c) 4 bits, and (d) 5 bits.

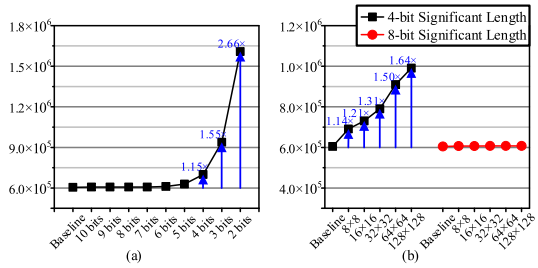

Fig. 18. Mean total losses of the proposed CNN calculated for the style of Composition VII in processing the COCO dataset by varying (a) the significant length and (b) the feature block size.

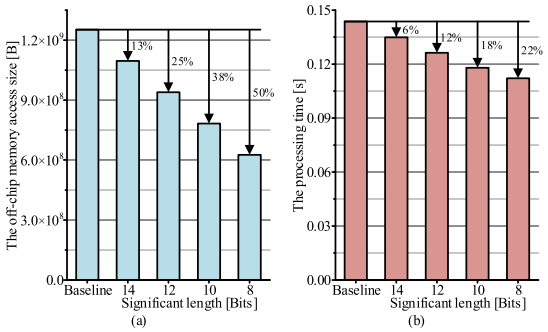

Fig. 19. Effects of significant length in the proposed data compression on (a) the off-chip memory access size and (b) the processing time.

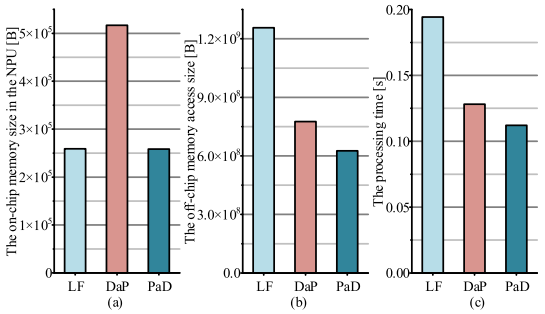

Fig. 20. Characteristics of three processing methods in processing an FHD image by the proposed CNN. (a) The on-chip memory size in the NPU, (b) the off-chip memory access size, and (c) the processing time.

The proposed data compression, there are two configuration factors: the feature block size and significant length. A feature block is configured to have the same size as the output size of the corresponding processing block so as to reduce the latency and the number of line buffers required, which is 1×64 for the vector processing block and 1×62 for the convolution processing block. To determine the significant length, a visual evaluation for the style of Composition VII is conducted by changing the significant length, and summarized in Fig. 17. The stylized image with a significant length of 2 bits is not sufficient to recognize the content, and so is the stylized image with a significant length of 3 bits. A significant length of 4 bits is at least required to recognize the content.

To evaluate the proposed data compression in depth, a quantitative evaluation has been performed by varying the significant length and the feature block size. The evaluation is summarized in Fig. 18, where the baseline has no compression and the mean total losses of the proposed CNN are calculated for the style of Composition VII in processing the COCO dataset. Fig. 18(a) shows that the mean total losses increase as the significant length decreases, and have a rapid increase for a significant length of 3 bits, which is coincident with the results in Fig. 17. Fig. 18(b) shows the mean total losses with significant lengths of 4 bits and 8 bits by chaining the feature block size. The mean total losses increase as the feature block size becomes larger with a significant length of 4 bits, while it is maintained with a significant length of 8 bits as 8 bits are large enough to store significant bits in a feature block.

Fig. 19 shows how the significant length in data compression affects the off-chip memory accesses and the processing time, where the baseline has a significant length of 16 bits. In case of off-chip memory accesses, the reduction is almost proportional to the difference between the word length and the significant length, as the feature maps is dominant in size and the total size of the shift amounts is quite small. The reduction of the processing time is smaller because the processing time depends on the computation time rather than the data transfer time in most layers. The 8-bit significant length seems to be a good choice as the reduction of the processing time is almost saturated at the length.

The layer-chaining technique is effective in reducing the off-chip memory accesses and improving the throughput further. Fig. 20 compares three processing methods in terms of on-chip memory size, off-chip memory access, and processing time when processing an FHD image by the proposed CNN, where LF denotes the layer-first processing [52], and DaP and PaD represent the depthwise-and-pointwise chaining [51] and the proposed pointwise-and-depthwise chaining, respectively. Fig. 20(a) shows the amount of on-chip memory realized in the NPU. The DaP chaining needs the largest buffer size as it stores a 3-D intermediate feature tile, while the LF and the PaD chaining need to store a 2-D intermediate feature tile. Fig. 20(b) summarizes the amount of off-chip memory accesses. Both the two layer-chaining methods have the smaller off-chip memory accesses than the LF, because the intermediate feature map produced by the front block is processed immediately by the rear block without accessing the off-chip memory. The PaD chaining has the smaller off-chip memory accesses than the DaP chaining, as the output feature map of the pointwise convolution is larger than that of the depthwise convolution. Fig. 20(c) shows the processing time. Since the size of off-chip memory accesses is reduced and the two convolutional layers are processed in parallel, the processing time of the layer-chaining methods is shorter than the LF. The PaD has the smaller processing time than the DaP due to the lower off-chip memory traffic.

TABLE VII
FPGA IMPLEMENTATIONS OF PIXEL-LEVEL LABELING CNNS

Architecture	[40]	[41]	[42]	[43]	[44]	[45]	[46]	This Work
FPGA	Xilinx XCKU040	Xilinx XC7K410T	Xilinx XCVU9P	Xilinx XCZU9EG	Xilinx XCVU13P	Xilinx XC7Z100	Intel Arria 10SX	Intel Arria 10 SoC
Application	Super-Resolution	Super-Resolution	Super-Resolution	Denoising	GAN	GAN	GAN	Style Transfer
Input Resolution	1920×1080 (FHD)	1440×640	1920×1080 (FHD)	3840×2160 (UHD)	n/a	n/a	n/a	1920×1080 (FHD)
Network Complexity [GMACs]	5.3	3.3	5.2	1018.2	n/a	n/a	n/a	37.6
Number of Layers	9	5	5	3	10	10	24	30
Architecture	Fully-Pipelined	Fully-Pipelined	Fully-Pipelined	Fully-Pipelined	Layer-First	Layer-First	Layer-First	Layer-Chaining
Compression	Yes	Yes	No	No	No	No	No	Yes
Quantization[a]	14-Bit for A 10-Bit for W	13-Bit for B	14-Bit for A 10-Bit for W	8-Bit for B	16-Bit for B	16-Bit for B	16-Bit for A 8-Bit for W	16-Bit for B
Clock Frequency [MHz]	150	130	200	150	190	200	200	200
DSP Blocks	1920	1512	2146	2493	1560	1987	576	1536
Adaptive Logic Elements[b]	151K LUTs 121K FFs	167K LUTs 158K FFs	94K LUTs 19K FFs	n/a	1325K LUTs	118K LUTs 247K FFs	n/a	189K ALMs
Block RAMs [KB]	194	945	53	n/a	11000	1978	n/a	3724
Frames Per Second	60	141	60	n/a	n/a	n/a	n/a	8.9
Throughput[c] [GMACs/sec]	318	390	312	294-348	n/a	n/a	176	335
Power Consumption [W]	5.69	5.38	6.91	n/a	n/a	n/a	n/a	5.90
Energy Efficiency [GMACs/sec/W]	55.93	72.49	45.14	n/a	n/a	n/a	n/a	56.76

[a]A, W, and B stand for activation, weight, and both activation and weight, respectively.
[b]The adaptive logic elements are lookup tables (LUTs) and flip-flops (FFs) for Xilinx FPGAs, and ALMs for Intel FPGAs.
[c]The throughput is calculated by multiplying the network complexity by the frames per second if not reported.

Table VII summarizes the characteristics of several FPGA implementations developed for pixel-level labeling applications. In [40], [41], [42], and [43], small-scale CNNs associated with a small number of layers are employed, so fully-pipelined architecture has been used to increase the throughput and reduce the off-chip memory accesses. In [44], [45], and [46], and this work, on the other hand, large-scale CNNs are used, so it is impossible to store all parameters and intermediate feature maps in the on-chip memory. The layer-first processing was used in [44], [45], and [46], while the layer-chaining technique is employed in this work. To reduce the computational complexity and the amount of parameters, this work has modified the network architecture and the kernel size, as was done in [40] and [41]. In addition, this work has applied the data compression technique to reduce the feature map size.

Our implementation needs more block RAMs than some previous implementations do, as the whole system including the display controller and the hard processor system wrapper has been realized. They take about 3.11MB and 524KB block RAMs that account for approximately 85% and 15% of the total block RAMs as shown in Fig. 14(c). To implement the NPU, about 67KB block RAM is used, which is smaller than or similar to the previous implementations. In addition, our implementation needs the smaller adaptive logic elements due to the smaller on-chip memory achieved by the proposed compression and the pointwise-and-depthwise chaining method.

The frames per second of this work is about 9, which is sufficient for image-to-image translation applications. The throughput of the proposed implementation, GMACs per second, is similar to those of the previous implementations realized in a fully-pipelined manner, and much higher than the previous GAN implementation. To compare energy efficiency, the power consumption of the proposed implementation has been estimated using a power analyzer in the Intel FPGA development toolsuit. The energy efficiency of the proposed implementation is higher than the previous implementations except for [41] targeting smaller input resolution.

VII. CONCLUSION

This paper has proposed a CNN inference accelerator for the style transfer applications, and realized it on a FPGA board. We modified StyleNet, a style transfer CNN, to reduce the computational complexity and the parameter size by making it have a lightweight network architecture and small kernels. The proposed CNN leads to almost no quality degradation compared to StyleNet. To reduce the feature map size, a new data compression method effective for the proposed CNN was developed with considering the computational complexity. In addition, a new layer-chaining method was presented to reduce the off-chip memory traffic and the overall processing time. Employing the proposed compression and layer-chaining techniques, the NPU equips with two processing blocks chained and run in parallel, and the compressor reduces the on-chip memory size and the off-chip memory traffic. A prototype CNN inference accelerator developed based on the proposed hardware architecture was implemented on an Intel Arria 10 SoC FPGA to process the proposed CNN. The implementation has achieved a throughput comparable to the previous FPGA implementations.

ACKNOWLEDGMENT

The authors would like to thank IC Design Education Center (IDEC), South Korea, for supporting the EDA tools.

REFERENCES

[1] 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.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–14.

[3] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[6] W. Liu et al., "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 21–37.

[7] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 694–711.

[8] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6924–6932.

[9] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1501–1510.

[10] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Universal style transfer via feature transforms," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1–11.

[11] Y. Li, M.-Y. Liu, X. Li, M.-H. Yang, and J. Kautz, "A closed-form solution to photorealistic image stylization," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 453–468.

[12] J. Yoo, Y. Uh, S. Chun, B. Kang, and J.-W. Ha, "Photorealistic style transfer via wavelet transforms," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9036–9045.

[13] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," 2015, *arXiv:1508.06576*.

[14] M. Ruder, A. Dosovitskiy, and T. Brox, "Artistic style transfer for videos," 2016, *arXiv:1604.08610*.

[15] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1125–1134.

[16] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2223–2232.

[17] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8789–8797.

[18] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 1857–1865.

[19] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," 2014, *arXiv:1501.00092*.

[20] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 391–407.

[21] W. Shi et al., "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.

[22] 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.

[23] C. Ledig et al., "Photo-realistic single image super-resolution using a generative adversarial network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4681–4690.

[24] 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. 136–144.

[25] P. O. Pinheiro and R. Collobert, "From image-level to pixel-level labeling with convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1713–1721.

[26] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels," in *Proc. IEEE Int. Conf. Embedded Softw. Syst. (ICESS)*, Jun. 2019, pp. 1–8.

[27] 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.

[28] J. Jo, S. Cha, D. Rho, and I.-C. Park, "DSIP: A scalable inference accelerator for convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 53, no. 2, pp. 605–618, Feb. 2018.

[29] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[30] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[31] J. Jo, S. Kim, and I.-C. Park, "Energy-efficient convolution architecture based on rescheduled dataflow," *IEEE Trans. Circuits Syst., Reg. Papers*, vol. 65, no. 12, pp. 4196–4207, Dec. 2018.

[32] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "ENVISION: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–248.

[33] S. Kim, J. Jo, and I.-C. Park, "Hybrid convolution architecture for energy-efficient deep neural network processing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 5, pp. 2017–2029, May 2021.

[34] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 218–220.

[35] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. 19th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2014, pp. 269–284.

[36] J. Yan, S. Yin, F. Tu, L. Liu, and S. Wei, "GNA: Reconfigurable and efficient architecture for generative network acceleration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2519–2529, Nov. 2018.

[37] H. Xu, Y. Wang, Y. Wang, J. Li, B. Liu, and Y. Han, "ACG-engine: An inference accelerator for content generative neural networks," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–7.

[38] Y.-Y. Hsieh, Y.-C. Lee, and C.-H. Yang, "A CycleGAN accelerator for unsupervised learning on mobile devices," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[39] Z. He, H. Huang, M. Jiang, Y. Bai, and G. Luo, "FPGA-based real-time super-resolution system for ultra high definition videos," in *Proc. IEEE 26th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2018, pp. 181–188.

[40] 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.

[41] J.-W. Chang, K.-W. Kang, and S.-J. Kang, "An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 1, pp. 281–295, Jan. 2020.

[42] S. Lee, S. Joo, H. K. Ahn, and S.-O. Jung, "CNN acceleration with hardware-efficient dataflow for super-resolution," *IEEE Access*, vol. 8, pp. 187754–187765, 2020.

[43] S. Colleman and M. Verhelst, "High-utilization, high-flexibility depth-first CNN coprocessor for image pixel processing on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 461–471, Mar. 2021.

[44] A. Yazdanbakhsh et al., "FlexiGAN: An end-to-end solution for FPGA acceleration of generative adversarial networks," in *Proc. IEEE 26th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2018, pp. 65–72.

[45] Y. Yu, T. Zhao, M. Wang, K. Wang, and L. He, "Uni-OPU: An FPGA-based uniform accelerator for convolutional and transposed convolutional networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 7, pp. 1545–1556, Jul. 2020.

[46] P. Yang, W. Mao, J. Lin, and Z. Wang, "A computation-efficient solution for acceleration of generative adversarial network," in *Proc. 18th IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2020, pp. 210–213.

[47] 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.

[48] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in *Proc. 54th ACM/IEEE Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.
[49] K. Goetschalckx and M. Verhelst, "Breaking high-resolution CNN bandwidth barriers with enhanced depth-first execution," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 323–331, Jun. 2019.
[50] M. Shi, P. Houshmand, L. Mei, and M. Verhelst, "Hardware-efficient residual neural network execution in line-buffer depth-first processing," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 4, pp. 690–700, Dec. 2021.
[51] B. Li et al., "Dynamic dataflow scheduling and computation mapping techniques for efficient depthwise separable convolution acceleration," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 8, pp. 3279–3292, Aug. 2021.
[52] S. Kim, S. Na, B. Y. Kong, J. Choi, and I.-C. Park, "Real-time SSDLite object detection on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 6, pp. 1192–1205, Jun. 2021.
[53] L. Bai, Y. Zhao, and X. Huang, "A CNN accelerator on FPGA using depthwise separable convolution," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 10, pp. 1415–1419, Oct. 2018.
[54] *Smart SSD*. Accessed: Nov. 20, 2022. [Online]. Available: https://semiconductor.samsung.com/ssd/smart-ssd/
[55] *Boards-and-Kits*. Accessed: Nov. 20, 2022. [Online]. Available: https://www.xilinx.com/products/boards-and-kits.html
[56] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[57] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[58] A. Howard et al., "Searching for MobileNetV3," 2019, *arXiv:1905.02244*.
[59] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016, pp. 1–14.
[60] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017, *arXiv:1710.09282*.
[61] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 393–405.
[62] J. Choi, B. Y. Kong, and I.-C. Park, "Retrain-less weight quantization for multiplier-less convolutional neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 3, pp. 972–982, Mar. 2020.
[63] P. Brown. (Nov. 2009). S3 texture compression, version 1.5. NVIDIA Corporation. [Online]. Available: http://www.opengl.org/registry/specs/EXT/texture_compression_s3tc.txt
[64] I. J. Goodfellow et al., "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
[65] *UltraScale Architecture and Product Data Sheet: Overview*. Accessed: Sep. 10, 2022. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ds890-ultrascale-overview
[66] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," 2014, *arXiv:1405.0312*.
[67] *7 Series DSP48E1 Slice*. Accessed: Sep. 10, 2022. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1
[68] *Intel Arria 10 Core Fabric and General Purpose I/Os Handbook*. Accessed: Sep. 10, 2022. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_dsp.pdf

Suchang Kim (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from Korea Aerospace University, Goyang, South Korea, in 2016, and the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2018, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering. His current research interests include VLSI architectures for neural network accelerators and computer arithmetic.

Boseon Jang (Graduate Student Member, IEEE) received the B.S. degree in electronic and electrical engineering from Hongik University, Seoul, South Korea, in 2021. He is currently pursuing the M.S. degree with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. His current research interests include VLSI architectures for error-correction codes, communication systems, and neural network processors.

Jaeyoung Lee (Graduate Student Member, IEEE) received the B.S. degree in electrical and electronics engineering from Konkuk University, Seoul, South Korea, in 2021. He is currently pursuing the M.S. degree with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. His current research interests include the algorithms and VLSI architectures for error-correcting codes, and deep neural networks.

Hyungjoon Bae (Graduate Student Member, IEEE) received the B.S. degree from the School of Electrical and Electronics Engineering, Chung-Ang University, Seoul, South Korea, in 2020, and the M.S. degree from the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2022, where he is currently pursuing the Ph.D. degree. His current research interests include high-speed VLSI architecture and computer architecture for general purpose microprocessor.

Hyejung Jang (Graduate Student Member, IEEE) received the B.S. degree in information systems and computer science from Hanyang University, Seoul, South Korea, in 2019. She is currently pursuing the M.S. degree with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. Her current research interests include VLSI architectures for error-correction codes, communication systems, and neural network processors.

In-Cheol Park (Senior Member, IEEE) received the B.S. degree in electronic engineering from Seoul National University, Seoul, South Korea, in 1986, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1988 and 1992, respectively.

Since June 1996, he has been an Assistant Professor with the School of Electrical Engineering, KAIST, where he is currently a Professor. Prior to joining KAIST, he was with the IBM T. J. Watson Research Center, Yorktown, NY, USA, from May 1995 to May 1996, where he researched high-speed circuit design. His current research interests include computer-aided design algorithms for high-level synthesis and very large scale integration architectures for general-purpose microprocessors.

Dr. Park received the Best Design Award at ASP-DAC in 1997 and the Best Paper Award at ICCD in 1999.