SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

February 2024, Vol. 67, Iss. 2, 122401:1–122401:14
https://doi.org/10.1007/s11432-022-3676-2

# FSS: algorithm and neural network accelerator for style transfer

Yi LING[1], Yujie HUANG[1,2], Yujie CAI[1,2], Zhaojie LI[1,2], Mingyu WANG[1*], Wenhong LI[1] & Xiaoyang ZENG[1]

[1]*State Key Laboratory of ASIC & System, Fudan University, Shanghai* 200120*, China;*
[2]*Shanghai ExploreX Technology Co., Ltd., Shanghai* 200120*, China*

Received 18 July 2022/Revised 21 October 2022/Accepted 5 January 2023/Published online 10 October 2023

**Abstract** Neural networks (NNs), owing to their impressive performance, have gradually begun to dominate multimedia processing. For resource-constrained and energy-sensitive mobile devices, an efficient NN accelerator is necessary. Style transfer is an important multimedia application. However, existing arbitrary style transfer networks are complex and not well supported by current NN accelerators, limiting their application on mobile devices. Moreover, the quality of style transfer needs improvement. Thus, we design the FastStyle system (FSS), where a novel algorithm and an NN accelerator are proposed for style transfer. In FSS, we first propose a novel arbitrary style transfer algorithm, FastStyle. We propose a light network that contributes to high quality and low computational complexity and a prior mechanism to avoid retraining when the style changes. Then, we redesign an NN accelerator for FastStyle by applying two improvements to the basic NVIDIA deep learning accelerator (NVDLA) architecture. First, a flexible dat FSM and wt FSM are redesigned to enable the original data path to perform other operations (including the GRAM operation) by software programming. Moreover, statistics and judgment logic are designed to utilize the continuity of a video stream and remove the data dependency in the instance normalization, which improves the accelerator performance by 18.6%. The experimental results demonstrate that the proposed FastStyle can achieve higher quality with a lower computational cost, making it more suitable for mobile devices. The proposed NN accelerator is implemented on the Xilinx VCU118 FPGA under a 180-MHz clock. Experimental results show that the accelerator can stylize 512×512-pixel video with 20 FPS, and the measured performance reaches up to 306.07 GOPS. The ASIC implementation in TSMC 28 nm achieves about 22 FPS in the case of a 720-p video.

**Keywords** neural network accelerator, style transfer, neural network, deep learning

## 1 Introduction

Multimedia fields, including photography, live broadcasts, and film and television productions, are enjoyed by many people. In recent years, neural networks (NNs) have shown their superiority in multimedia processing, including in super-resolution [1–4], denoising [5–7], and deblurring [4, 8, 9]. With the help of an NN, style transfer, which tries to migrate any style to an image or video while maintaining its content [10], has broken through the simple filter function and become a popular multimedia application.

Owing to the popularity of mobile devices, such as cameras and smartphones, multimedia must be able to run on these devices [11]. Because mobile devices are energy-sensitive, the circuits and chips are usually resource-constrained. Thus, instead of a powerful server GPU, an efficient NN accelerator is necessary to run NN algorithms [11, 12] on mobile devices.

Then, the NN algorithms run on mobile devices are expected to have low computational complexity. However, existing style transfer networks are complex, limiting the style transfer applied on mobile devices to a finite set of styles and non-real-time applications. The limited style transfer on mobile devices cannot satisfy people's growing desire to switch arbitrary styles for images and real-time videos. Thus, in this paper, we design a system named FastStyle system (FSS), where a novel algorithm and general NN accelerator are proposed for arbitrary style transfer.

* Corresponding author (email: mywang@fudan.edu.cn)

 

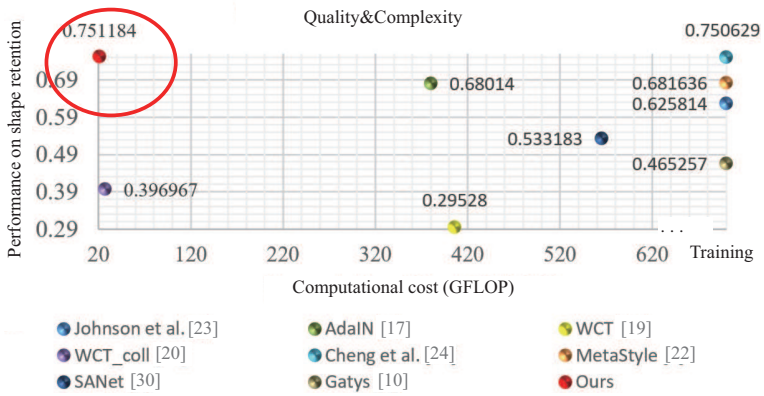


**Figure 1** (Color online) The performance on shape retention (higher is better) and floating point operation amounts of different algorithms. GFLOP refers to giga floating-point operations.

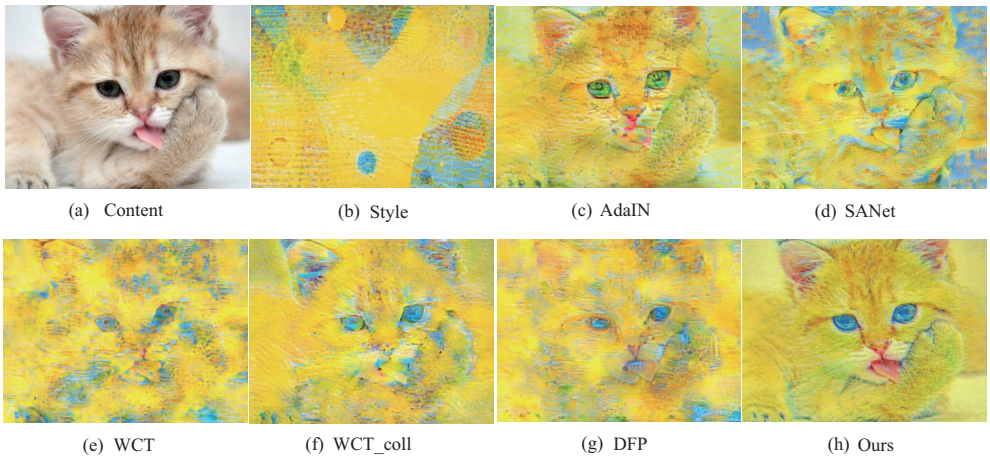


**Figure 2** (Color online) Distortion is visible in the images generated by other state-of-the-art (SOTA) algorithms ((c) AdaIN [17], (d) SANet [18], (e) WCT [19], (f) WCT_coll [20], (g) DFP [21]), but not by (h) ours. (a) and (b) Content and style images, respectively.

Since Gatys et al. [10] introduced the NN to style transfer, NN has entered a brand new era. Much research has been conducted on NN-based style transfer, including photorealistic style transfer [13], stroke size [14, 15], photo compositing [16], and significant arbitrary style transfer [17–20].

Nevertheless, as shown in Figure 1 [17–21], in arbitrary style transfer, the SOTA algorithms either achieve unsatisfactory performance in shape retention [17–20] or require time-consuming retraining when the style changes [10, 22–24]. Here, we use the structural comparison function in SSIM [25] to measure the performance in shape retention, similar to what is done in [24]. As shown in Figure 2, serious distortions are visible in the images generated by AdaIN [17], SANet [18], WCT [19], WCT_coll [20], and DFP [21]. Moreover, except for WCT_coll, all other algorithms' computational complexity is huge, and their real-time performance relies on an Nvidia TITAN GPU. Neither retraining nor the TITAN GPU is suitable for resource-constrained circuits or chips on energy-sensitive mobile devices. Although knowledge distillation for style transfer is proposed in WCT_coll to reduce the NN complexity, it is still subject to content distortion.

As a result, we propose a novel arbitrary style transfer algorithm: FastStyle. In [17–21], retraining is avoided when the style changes, but distortion is caused by changing the data distribution of the content feature map according to the style feature map. Moreover, high-quality stylized images are generated through a feed-forward network, and various styles correspond to different weights of the network [22–24, 26], in that they need to be retrained when the style changes. Therefore, in FastStyle, we propose a light network to transfer the content image to a specific style, contributing to high quality and a low computational cost. Furthermore, we present the priori mechanism to dynamically adjust the weights according to the style image (priori information), avoiding retraining when the style changes. As shown in Figures 1 (red circle) and 2, FastStyle achieves higher quality with a lower computational cost.

Then, an NN accelerator adaptive to the FastStyle is necessary for mobile devices. Recent studies have focused on designing NN accelerators to make inferences with higher energy efficiency and lower latency [27–30]. The power limitation in mobile devices excludes GPU-based accelerators from consideration. Importantly, FPGA can achieve higher energy efficiency than GPU [31, 32] and is therefore an indispensable platform for implementing the NN accelerator for mobile devices. Moreover, as a reconfigurable and reprogrammable device, FPGA is more flexible than ASIC. Through FPGA, researchers can design NN accelerators oriented to one specific network to further optimize the performance of the accelerator in particular aspects, such as latency, power consumption, and energy efficiency [28,31,33,34], which is suitable for the diversity of multimedia applications [12].

However, most current FPGA-based accelerators focus on NNs for high-level computer vision tasks, such as classification and detection. Generally, these NNs consist of convolutional layers, pooling layers, fully connected layers, batch normalization layers, and residual layers [35, 36]. Unlike them, the instance normalization layer [37] and Gram matrix [10] are widely used in the networks of style transfer. If directly mapping the FastStyle to existing NN accelerators, the data dependency in the instance normalization layer and the lack of acceleration for the Gram operation limit the performance.

NVDLA[1)] is an efficient open-source NN accelerator provided by NVIDIA. We design a general NN accelerator based on NVDLA to adapt to FastStyle. All layers of FastStyle, including the Gram matrix, are integrated into a dedicated designed data path after optimization. In addition, the correlation between video frames is utilized to remove the data dependency in the instance normalization, which improves the accelerator performance by 18.6%.

The main contributions of this paper are as follows.

• We design a novel system named FSS to achieve real-time arbitrary style transfer, which can also efficiently run general NNs. FSS presents innovations in both the algorithm and hardware.

**Algorithm:**

• We propose a novel arbitrary style transfer algorithm named FastStyle, which achieves higher-quality results with a lower computational cost.

• Moreover, we propose a new mechanism to dynamically adjust the network weights, avoiding retraining when the style changes.

**Hardware:**

• To the best of our knowledge, this is the first study to deploy a style transfer network on FPGA that can explore the continuity of the video stream. Redesigned FSMs are added to NVDLA to increase the flexibility of NVDLA; thus, convolution and the Gram operation share the same processing element array in the accelerator, enabling our accelerator to achieve high DSP utilization.

• We propose a method to detect the scene change and simplify the instance normalization operation based on the continuity of the video stream, which results in an 18.6% performance improvement.

## 2 Related work

### 2.1 Arbitrary style transfer

Deep learning arbitrary style transfer was started by Gatys et al. [10], who used the feature maps extracted by VGG16 [35] to construct the content and style loss, and train the white noise image according to the loss. Because the training process is time-consuming, one feed-forward network was proposed to transfer one or several specific styles in [23, 26, 38–40]. As such, once the style is changed, a long period is required for retraining the algorithm. MetaStyle [22] applies bi-level optimization of meta learning on style transfer, which requires 200 updates to adapt to a new style. AdaIN [17] proposes adaptive instance normalization for arbitrary style transfer. In SANet [18], instead of matching between the second-order statistics of features, a style-attention network is proposed. Except for the above algorithms, patch-based ways are also used in arbitrary style transfer [19,41], but they are usually computationally expensive. The shape retention or computational complexity of the above algorithms is not satisfactory. The proposed FastStyle achieves high quality with low computational complexity.

1) NVDLA. http://nvdla.org.

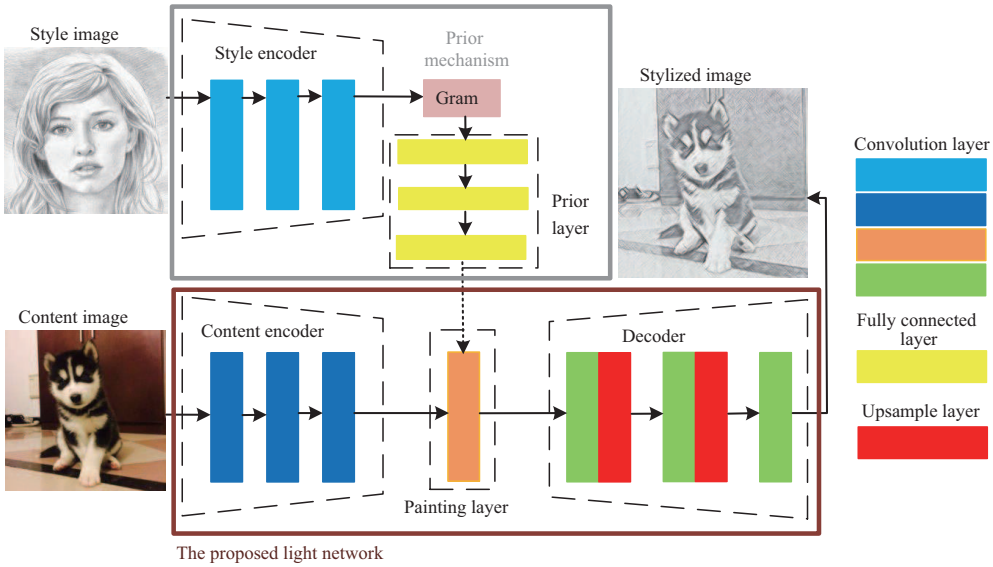


**Figure 3** (Color online) The architecture of the proposed FastStyle. The solid arrows indicate the data flow, and the dotted arrows indicate the parameter setting. The light network is responsible for transferring the content image to a specific style. The prior mechanism is used to dynamically adjust the weights of the Painting Layer based on the style image without retraining.

### 2.2 Neural network accelerator

Recently, studies have focused on FPGA-based NN accelerators. The authors of [30] proposed a high-performance specific NN accelerator on FPGA. The utilization of a layer-wise ping-pong buffer improves the throughput, and batch-based computing in the fully-connect layer creates more opportunities for data reuse. The authors of [27] implemented VGG16 on FPGA with dynamic data precision, eliminating the need for memory bandwidth. An architecture [42] was presented to adapt to layers with diverse dimensions and sizes in deep residual NNs like ResNet. Some FPGA-based NN accelerators are dedicated to implementing fast NN algorithms [33] and exploring sparsity in convolution neural networks [32]. However, the commonly used instance normalization and Gram matrix [10] in style transfer limit the performance of the above accelerators. Although PLACID was proposed in [12] to automatically create the corresponding efficient FPGA-based NN accelerators according to the specific NNs, a general accelerator applied on mobile devices for diverse applications cannot be generated.

Moreover, NVDLA is a free and open-source architecture for IoT devices, with low power consumption. Accordingly, we design a general NN accelerator to adapt to the FastStyle network, which can run with high power efficiency.

## 3 The proposed algorithm

The architecture of the proposed FastStyle is shown in Figure 3. The proposed light network can transfer the content image to a specific style. Furthermore, the proposed prior mechanism is used to dynamically adjust the weights of the Painting Layer based on the style image (prior information), making the light network adapt to a new style without retraining. The process of arbitrary style transfer for the image is as follows:

(1) The feature maps of the style and content images are extracted by the Style and Content Encoder, respectively;

(2) The Gram matrix of the feature map of the style image is sent to the Prior Layer to generate the weights of the Painting Layer;

(3) The Painting Layer works on the feature map of the content image, and then its output is fed into the Decoder;

(4) The Decoder maps the feature map into the stylized image.

To avoid complex retraining when the style changes, we propose the prior mechanism where the weights of the Painting Layer are dynamically generated by the Prior Layer based on the style image (prior information) to adapt to the new style.

When FastStyle is applied to video, all the frames are transferred into a fixed style. The Style Encoder and Prior Layer work first to initialize the weights of the Painting Layer based on the style image; then only the Content Encoder, Painting Layer, and Decoder (the proposed light network) work to continuously migrate the style for the video frames.

### 3.1 Style and content encoder

The Style and Content Encoder consist of three convolution layers: Conv_3_32_1, Conv_3_64_2, and Conv_3_128_2 (from left to right in Figure 3), where Conv_$A$_$B$_$C$ means a convolution layer with the kernel size $A\times A$, $B$ channels, and stride $C$. Each convolution layer in the Style Encoder is followed by a ReLU activation, while each one in the Content Encoder is followed by an instance normalization and a ReLU activation because instance normalization helps eliminate the style in the original content image [17]. Instance normalization is calculated as

$$F_i^c = \frac{F_i^c - \mu^c}{\sigma^c + \varepsilon}, \tag{1}$$

where $F_i^c$ means the $i$th point in the $c$th channel of the feature map $F$. $\mu^c$ and $\sigma^c$ are the mean and standard deviation, respectively, of the $c$th channel in the feature map $F$. $\varepsilon$ is used to prevent the denominator from being zero.

### 3.2 Prior mechanism and layer

Although high quality is achieved, the retraining or additional updates when the style changes come with a high computational cost [22–24, 26]. Thus, we propose that the prior mechanism adapts to the new style without retraining.

In the prior mechanism, we design the Prior Layer to dynamically generate the weights of the Painting Layer according to the style image (prior information). The Prior Layer desires the style information, so we further extract the style information from the style feature maps with a Gram operation that can represent the style of one image [10]. Each value in the Gram matrix of a feature map can be calculated as

$$g_{ij} = \sum_{k=0}^{N-1} f_k^i f_k^j, \tag{2}$$

where $f_k^i$ means the $k$th value in the $i$th channel of the feature map, and $N$ is the product of the width and height of the feature map. Then, the Gram matrix of the style feature map is used as the input of the Prior Layer.

The Prior Layer consists of three fully connected layers whose input and output amounts are ($128\times128$, 2000), (2000, 2000), and (2000, $3\times3\times128\times128$) (from top to bottom in Figure 3), respectively. Each fully connected layer except the last one is followed by a ReLU activation.

### 3.3 Painting Layer

The Painting Layer is responsible for stylizing the content feature map according to the style feature map.

The weights of the Painting Layer are dynamically generated by the Prior Layer based on the style image to adapt to the new style, and its input is the feature map of the content image. The Painting Layer consists of Conv_3_128_1 followed by a ReLU activation, which is lightweight. The outputted stylized feature map is fed into the Decoder.

### 3.4 Decoder

The Decoder is symmetrical to the Content Encoder, so it consists of Conv_3_64_1, Upsample_2, Conv_3_32_1, Upsample_2, and Conv_3_3_1 (from left to right in Figure 3), where Upsample_2 means double upsampling with the nearest neighbor interpolation. Each convolution layer is followed by a ReLU activation function except the last one. Then, the stylized image is achieved as

$$\mathrm{out}_{xy}^l = 255\frac{f_{xy}^l - f_{\min}^l}{f_{\max}^l - f_{\min}^l}, \tag{3}$$

where $f^l_{xy}$ means the value at point $(x, y)$ in the $l$th channel of the output of the Decoder, and $f^l_{\min}$ means the minimum value in the $l$th channel while $f^l_{\max}$ means the maximum.

### 3.5 Loss function

The loss function consists of content loss and style loss:

$$\text{loss} = w_c \cdot \text{loss}_{\text{content}} + w_s \cdot \text{loss}_{\text{style}}, \tag{4}$$

where $w_c$ and $w_s$ are the weight of the content and style loss, respectively. In this paper, $w_c$ is 1, and $w_s$ is 5. The content and style loss are similar to those in [17],

$$\text{loss}_{\text{content}} = ||\varphi(I_c) - \varphi(I_g)||_2^2, \tag{5}$$

$$\begin{aligned}\text{loss}_{\text{style}} = \sum_{i=1}^{L}(&||\mu(\varphi_i(I_s)) - \mu(\varphi_i(I_g))|_2^2 \\ &+ ||\sigma(\varphi_i(I_s)) - \sigma(\varphi_i(I_g))|_2^2),\end{aligned} \tag{6}$$

where $\varphi$ means the feature map outputted by VGG19 [35]; $\mu$ represents calculating the channel-wise mean of the feature map; $\sigma$ means the standard deviation; and $I_c$, $I_s$, and $I_g$ refer to the content, style, and generated image, respectively. We use the feature map outputted by relu3_4 of VGG19 in the content loss and the feature maps outputted by relu1_2, relu2_2, relu3_4, and relu4_4 in VGG19 in the style loss.

### 3.6 Training details

The proposed FastStyle is trained end to end on the ILSVRC2012 [43] validation set, WikiArt[2)] training set, and DTD [44] dataset. The ILSVRC2012 validation set contains 50k images, which are used as content images. Furthermore, 80k images in the WikiArt training set and 5057 images in the DTD are used as the style images. During training, all the images are resized to 300×300, and one content and style image is randomly selected in each iteration. We apply the Adam [45] with a fixed learning rate of 0.0001 for 1.6 million iterations. The FastStyle is implemented with Tensorflow [46], which takes about 36 h for the training on an Nvidia 1080ti GPU. During training, the 32-bit floating-point number is utilized in the GPU. Then, the network is quantified to the 16-bit fixed-point number to run on the accelerator.

## 4 Hardware design

The difference between FastStyle and the general NNs is that there are Gram operations and instance normalization in FastStyle. Inefficient operations in TCONV layers [47] challenge the hardware acceleration, and this problem is addressed by the computation reformulation in [47]. Unlike [47], this work proposes a novel algorithm to avoid TCONV layers, so we do not have a penalty of TCONV layers. Therefore, we design an NN accelerator based on the open-source NVDLA, which is used to accelerate general NNs.

### 4.1 Hardware overview

NVDLA maps convolution and a fully connected layer to a fixed-size matrix-vector multiplier in an efficient way, where convolution and the fully connected layer share a highly efficient data path.

The data path of NVDLA is composed of the modules without fill color in Figure 4. A 2-megabyte (MB) local buffer is utilized as a local warehouse to send data to the data path with high bandwidth and low read latency. An efficient direct memory access (DMA) engine exchanges data between the external memory and local buffer. The finite state machines of fetching data and weights (dat FSM and wt FSM) continuously send a read request to the local buffer. The requested data and weights are sent to the convolutional multiply-accumulator (CMAC). CMAC consists of $P_k \times P_c$ multiply accumulation (MAC) units. In a clock cycle, a $P_k \times P_c$ matrix multiplication is performed, where $P_k$ and $P_c$ are the parallelisms of the output and input channel, respectively. The outputs of the CMAC are accumulated

2) K. nicol painter by numbers WikiArt. 2016. https://www.kaggle.com/c/painter-by-numbers.

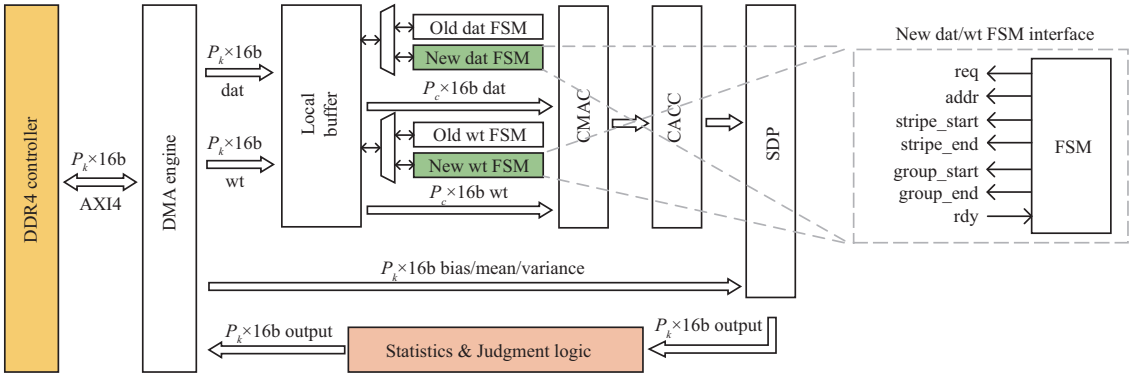


**Figure 4** (Color online) The proposed hardware structure. First, the data are fetched into the local buffer and then sent to the convolutional multiply-accumulator (CMAC). After accumulation in the convolutional accumulator (CACC), the outputs are written back to the external memory. The data flow is mainly controlled by the finite state machines of fetching data and weights (dat FSM and wt FSM). Modules without fill color are the original data path of NVDLA. A new dat FSM and a new wt FSM are redesigned to bypass the original datFSM and weight FSM, which gives the whole architecture the ability to perform other operations (including GRAM operation) by software programming. 'Statistics' and 'Judgment logic' are responsible for obtaining the mean and variance and detecting the scene changes.

in the convolutional accumulator (CACC), and the outputs of CACC are sent to the single data point processor (SDP) that works in the stream mode and is used for bias-add, instance normalization, and upsampling. The output of SDP is written back to external memory and becomes the input of the next layer. To adapt to FastStyle, we first redesign a new dat FSM and a new wt FSM to bypass the original dat FSM and weight FSM. The new FSMs give the whole architecture the ability to perform other operations (including GRAM operation) by software programming. Then, after analyzing the change of mean and variance with the frame change, we find that it is possible to remove data dependency in instance normalization owing to the continuity of the video stream. Statistics and judgment logic are added to the data path of NVDLA, which fetches the data needed by instance normalization and detects the scene change. The overall architecture of the accelerator is shown in Figure 4.

### 4.2 Adding flexible FSMs to enable NVDLA to perform other operations except convolution

The dat FSM and wt FSM of NVDLA only support the data flow of convolution operations (and the fully connected layer). The NVDLA runs convolution operations by the step of four kinds of operations, as follows.

(1) Atomic operation: This is calculated in a single cycle, where feature data are broadcasting to all the kernels, and each kernel has a single partial sum result;

(2) Stripe operation: This contains several atomic operations, all of which share the same weight data. The first atomic operation of a stripe operation is marked as "stripe start", and the last atomic operation is marked as "stripe end". Stripe operation produces a series of partial sum results for each kernel;

(3) Block operation: This contains several stripe operations, each of which contributes the same output in a block. The partial sum results of each stripe operation are accumulated in the module CACC;

(4) Group operation: This contains several block operations, and after a group operation, the final accumulator result in the module CACC is ready for output. The first block operation is marked as "group start", and the last is marked as "group end". The module CACC sends the accumulated results out after the last block operation.

Therefore, we redesign a new dat FSM and a new wt FSM to bypass the original dat FSM and weight FSM, as in Figure 4. The new dat FSM and wt FSM both contain a flexible read request generation engine. The pattern of the requested data address and the size of the stripe operation, block operation, and group operation of the new FSMs can all be programmed. The new dat FSM and new wt FSM logic enable the whole architecture to perform other operations except convolution.

The Gram operation is widely used in style transfer networks and is realized with matrix multiplication. The input feature is reshaped into a two-dimensional matrix whose width is $W$ and height is $H$. The GRAM operation is done by setting the stripe length to $P_k$ atomic operations, setting the block length to ceil($H/P_c$) stripe operations, and setting the group length to one block operation, in which ceil() means rounding up. Thus, we ensure that each atomic operation finishes the computation of $P_k \times P_c$

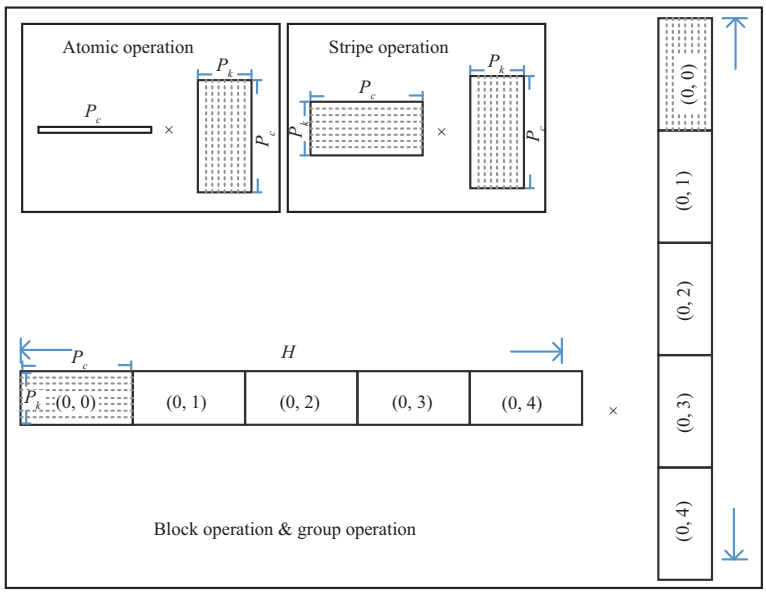


**Figure 5** (Color online) The Gram operation is realized by a matrix multiplication and can be decomposed into ceil($W/P_k$) ×ceil($W/P_k$) group operations. Each group operation contains one block operation, and each block operation contains ceil($H/P_c$) stripe operations. Each stripe operation contains $P_k$ atomic operations and finishes the $P_k$ computations of $P_k \times P_c$ multiplications.

multiplication, and each stripe operation finishes the $P_k$ computations of $P_k \times P_c$ multiplications. After ceil($H/P_c$) stripe operations, the accumulated results in the module CACC are sent out, as in Figure 5. The whole Gram operation can be accomplished by ceil($W/P_k$)×ceil($W/P_k$) Group operations.

### 4.3 Instance normalization optimization based on scene detection

The instance normalization calculates the channel-wise mean and variance of the input first, where the data dependence on the statistical information of the input prevents the instance normalization operation from starting before the operation of the previous layer is complete. The input of the instance normalization comes from the convolution in FastStyle, and the convolution output is saved to external memory and then read back to start the instance normalization because of the limited local buffer size. This kind of data flow increases the access to the external memory, leading to more power consumption and a more severe memory bottleneck. Furthermore, the arithmetic unit is also halted until the instance normalization operation finishes, dropping the efficiency. To solve this problem, researchers proposed hardware-aware mathematical transformation in [48] for image-to-image translation. Unlike the method proposed in [48], our proposed method detects the scene change, which eliminates the off-chip memory traffic of the instance normalization layer by exploring the continuity of the video stream.

The experiments demonstrate that the change in mean and variance is minimal between frames in the same scene, and a drastic change only happens when the scene changes as in Figure 6. Thus, we propose to detect the scene change based on the mean and variance differences. Accordingly, we propose to implement the instance normalization with the mean and variance calculated from the previous frame in the same scene, and rectify the result by applying a channel-wise linear operation when the scene changes to prevent the image distortion:

$$y' = \frac{x - a'}{b'}, \quad y'' = \frac{x - a''}{b''} \Rightarrow y'' = \frac{y' - \frac{a'' - a'}{b'}}{\frac{b''}{b'}}, \tag{7}$$

where $x$ is the input of instance normalization; $a'$ and $b'$ are the channel-wise mean and standard deviation of the previous frame, respectively; $y'$ is the output before rectification; $a''$ and $b''$ are the channel-wise mean and standard deviation of current frame, respectively; and $y''$ is the rectified output.

Then, when the scene is unchanged, the instance normalization operation can be realized in the data flow of the previous layer, eliminating extra external memory access. Moreover, the arithmetic units are not halted. To achieve this, we add statistics and judgment logic to the data path to obtain the mean and variance and then detect the scene changes.

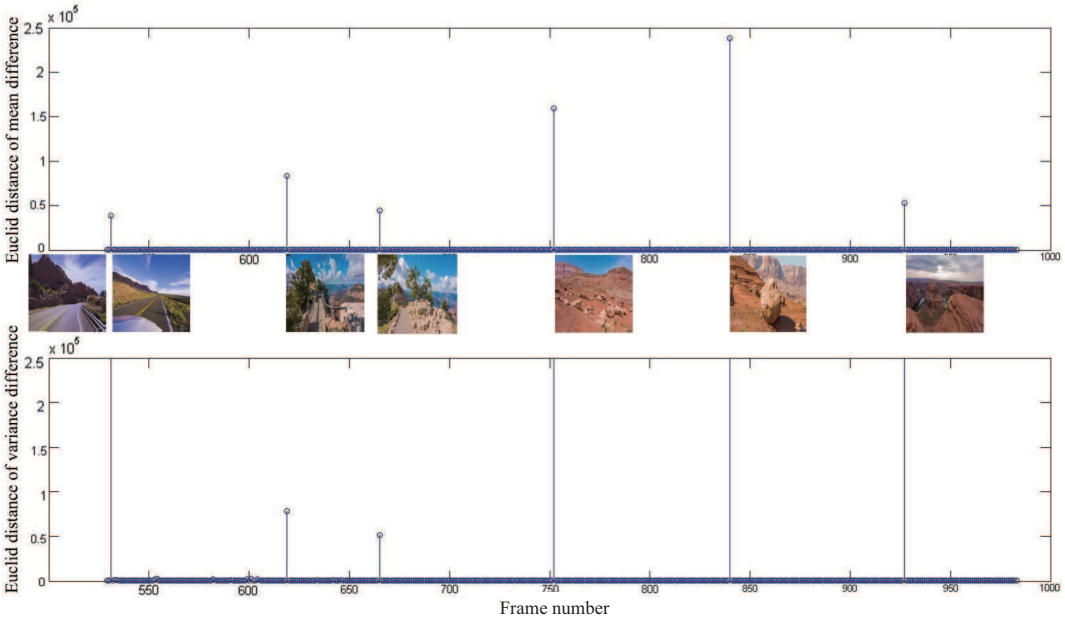


**Figure 6** (Color online) The image shows the change in the mean and variance with the frame change. The $x$-axis is the frame number, and the $y$-axis is the Euclid distance of the mean and variance vectors of two adjacent frames. The difference between the mean and variance is minimal between frames in the same scene, so the mean and variance of the previous frame can replace that of the current frame to perform instance normalization when the scene is not changed. The apparent peaks indicate the frame numbers of the scene change.

## 5 Experimental results

In this section, we present the experimental results from both the algorithm and hardware aspects to evaluate the proposed FSS system.

### 5.1 Algorithm evaluation

**Qualitative and quantitative comparison.** We qualitatively and quantitatively compare the proposed algorithm FastStyle with the recent SOTA arbitrary style transfer algorithm AdaIN [17], SANet [18], and MetaStyle [22]. As shown in Figure 7, the lines are distorted in the images generated by AdaIN, which makes the contents in the images look strange. For example, the generated image by AdaIN is distorted in the fifth row in Figure 7 when the style and content images are all photorealistic. Also, in the third and fourth rows in Figure 7, the distortion of the images generated by AdaIN is also severe. Owing to the matching process in the SANet, the content in the style image causes severe distortion in the images generated by SANet, as in the eyes in the first row, large deformation in the fourth row, and the ghosts in the fifth row. Color blocks in the images generated by MetaStyle undermine the integrity and aesthetics of the image content, as in the third and fourth rows.

The proposed FastStyle achieves good performance in shape retention and style migration, in addition to distinguishing between style and content in the style image, and reasonably distributing the color in the style image to the generated image. Furthermore, the good shape retention of FastStyle results in potential application to photorealistic images, as in the fifth row. Moreover, as shown in Figure 1, the quantitative comparison shows that the FastStyle achieves higher performance in shape retention than the SOTA algorithms.

**User study.** In the user study, 400 stylized images are generated by 20 content images and 20 style images, and 50 volunteers are invited to score them. Each volunteer conducts 10 tests where they are asked to score the generated images obtained by different algorithms with [1 (most disliked), 2, 3, 4 (favorite)]. Then, the favorite rate (the ratio of the number of times each algorithm obtains the highest score to the total number of tests) and average score (the ratio of the total score obtained for each algorithm to the total number of tests) of each algorithm are obtained. Here, the total number of tests is 500 (50×10). As shown in Figure 8, the proposed FastStyle achieves the highest favorite rate and average score.


**Figure 7** (Color online) The visual results of the algorithms. We recommend zooming in and looking at the color version.

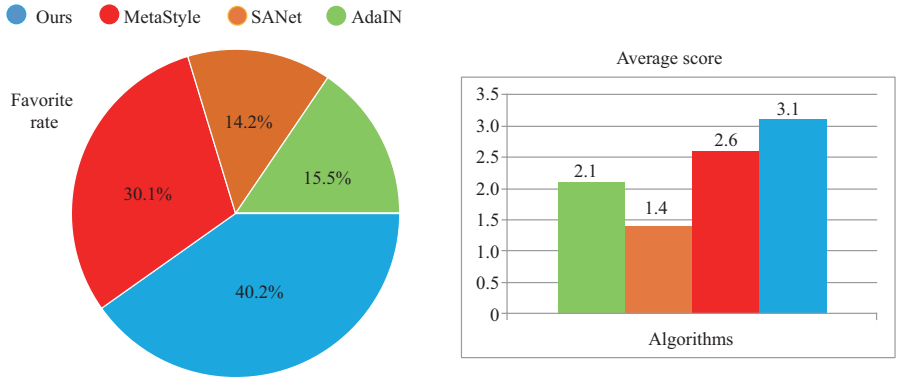


**Figure 8** (Color online) The favorite rates (the ratio of the number of times each algorithm with the highest score to the total number of tests) and average scores (the ratio of the total score obtained for each algorithm to the total number of tests) of different algorithms through a user study.

**Table 1** Floating-point operation amounts of different algorithms. The floating-point operation amounts ($10^9$) to different algorithms in the case of unchanged style and 512×512-pixels input images. Ours is the only algorithm with its customized hardware implementation.

| Algorithm | AdaIN [17] | SANet [18] | MetaStyle [22] | Ours |
|---|---|---|---|---|
| $10^9$ | 253.1 | 368.5 | 77.5 | 15.4 |
| Hardware | No | No | No | Yes |

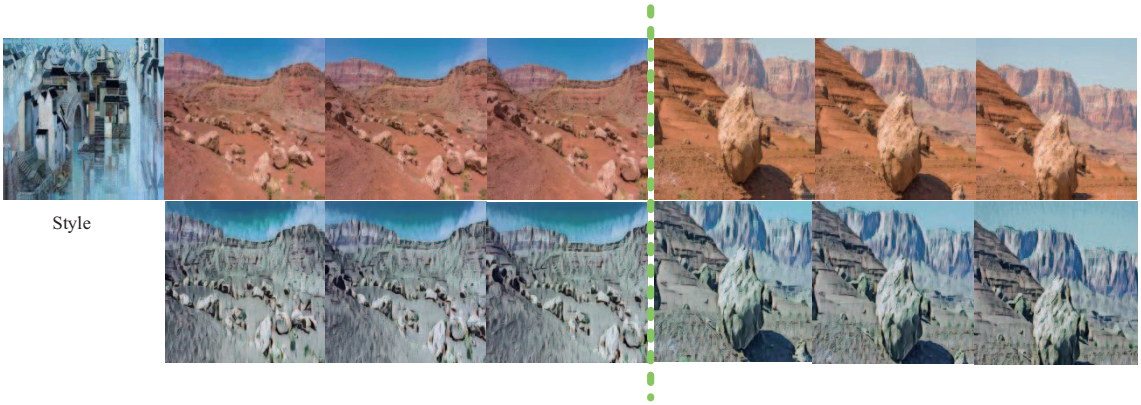


**Figure 9** (Color online) The visual result of the proposed system applied on a specific style transfer for video. The style image is on the top left. The images in the first row are the original frames in the video, and the ones in the second row are their corresponding stylized results. The green dotted line represents a scene change.

**Table 2** The resource utilization of the proposed accelerator on Xilinx FPGA board VCU118 with 180 MHz

| Resource | Used | Available | Utilization (%) |
|---|---|---|---|
| LUT | 152397 | 1182240 | 12.89 |
| LUT_RAM | 2512 | 591840 | 0.42 |
| FF | 100146 | 2364480 | 4.24 |
| BRAM | 493.50 | 2160 | 22.85 |
| DSP | 2134 | 31.20 | 31.20 |

**Comparison on computational cost.** We compare the floating-point operation amounts of the different algorithms when the input style and content images are 512×512 pixels in two cases: where the style is constantly changed and unchanged. As shown in Table 1, in the case of unchanged style (for video), except for the proposed FastStyle, MetaStyle has the least floating-point operation amount, but it is about five times that of FastStyle. As shown in Figure 1, in the case of constantly changing style, the algorithms proposed in [10, 22–24] need retraining to adapt to a new style (not suitable for mobile devices), while in [17–20] our FastStyle does not need retraining. And the FastStyle achieves the minimum computational cost, which is 22.2% lower than WCT_coll. As a result, the proposed FastStyle is more suitable for mobile devices than the SOTA algorithms. Furthermore, the FastStyle is the only one that has its own hardware implementation.

### 5.2 The performance of the accelerator

We implement the accelerator on Xilinx FPGA board VCU118 with 180 MHz. The result generated by the accelerator for a video is shown in Figure 9, and it can be found that the optimization on the instance normalization can work well when the scene changes. The measured performance of accelerator reaches up to 306.07 GOPS when running our FastStyle algorithm, and the resource utilization is shown in Table 2. The CMAC with a size 64×32 consumes 2048 DSP resources.

As shown in Table 3, the proposed accelerator reaches about 20 FPS when performing style transfer on a 512×512 video stream, and 85.37 ms is needed to adapt to the new style.

### 5.3 Evaluation of the Gram layer in the accelerator

As there are no other hardware implementations on the gram layer, we choose to evaluate our design against a software implementation. We implement a *C* version of the Gram operation in OpenMP. The

**Table 3** Performance of the accelerator on FastStyle[a)]

| | Structure | Cycles | Time (ms) | Total (ms) |
|---|---|---|---|---|
| CE1 | CONV+INORM | 2.38 M | 13.22 | |
| CE2 | CONV+INORM | 1.20 M | 6.67 | |
| CE3 | CONV+INORM | 647.22 k | 3.60 | |
| PL | CONV | 1.23 M | 6.83 | 50.42 |
| D1 | CONV+upsample | 640.49 k | 3.56 | |
| D2 | CONV+upsample | 597.11 k | 3.32 | |
| D3 | CONV+upsample | 2.38 M | 13.22 | |
| SE1 | CONV | 2.38 M | 13.22 | |
| SE2 | CONV | 1.20 M | 6.67 | |
| SE3 | CONV | 647.22 k | 3.60 | |
| GRAM | GRAM | 195.49 k | 1.09 | 85.37 |
| FC1 | FC | 1.07 M | 5.94 | |
| FC2 | FC | 132.34 k | 0.74 | |
| FC3 | FC | 9.74 M | 54.11 | |

a) CE means the convolution layer of the Content Encoder; PL means the Painting Layer; D means the convolution layer of the Decoder; SE means the convolution layer of the Style Encoder; Gram means calculating the Gram matrix of the feature map; FC is the fully connected layer in the Prior Layer.

**Table 4** Performance on Gram layer

| Device | Frequency | Data type | Time (ms) |
|---|---|---|---|
| Xilinx VCU118 (ours) | 180 MHz | Int16 | 1.09 |
| Intel i7-7700K 4C8T | 4.20 GHz | Float | 41.94 |
| | | Int16 | 20.96 |
| AMD Ryzen 7 1700 8C16T | 2.80 GHz | Float | 30.61 |
| | | Int16 | 15.26 |
| Arm A53 2C2T | 650 MHz | Float | 1068.56 |
| | | Int16 | 915.35 |

**Table 5** Performance comparison between with and without instance normalization optimization

| | Time before optimized (ms) | Time after optimized (ms) |
|---|---|---|
| Conv1 | 16.28 | 13.22 |
| Conv2 | 8.22 | 6.67 |
| Conv3 | 4.36 | 3.60 |

$C$ code is compiled with gcc in $O_3$ optimization mode. The results in Table 4 verify that the accelerator outperforms mainstream CPUs when running the Gram layer.

### 5.4 Evaluation of the optimization on instance normalization

A total of 224 mean and variance vectors need to be computed for each frame because of instance normalization. We propose to use the change in mean and variance to detect the scene changes. As shown in Figure 6, apparent peaks can be found when the scene changes, and the difference in the mean and variance is minimal between frames in the same scene. Therefore, we use the mean and variance of the previous frame to perform instance normalization when the difference is smaller than a specific threshold. Otherwise, a channel-wise linear operation, as in 7, is performed to rectify the output.

We evaluate the inference time of convolution layers following the instance normalization layer before and after optimization. As shown in Table 5, an 18.6% performance improvement is gained after the optimization, where the CMAC unit is not halted owing to external memory access when the scene is not changed.

### 5.5 The ASIC implementation

The proposed accelerator is also implemented with the TSMC 28-nm HVT process. As shown in Table 6, the logic area is 2.69 $mm^2$, and the power is 1.28 W. Besides, we evaluate the speed on a 720-p video by RTL simulation, where the DRAM behavior model is provided by Micron, and 21.67 FPS is reached.

**Table 6** The ASIC implementation result of the accelerator

| Process | TSMC 28 nm HVT CMOS |
|---|---|
| Logic gate count | 2896943 |
| Logic area | 2.69 $mm^2$ |
| Frequency | 1 GHz |
| Voltage | 0.9 V |
| Power | 1280 mW |
| Leakage power | 861.28 μW |
| DRAM bandwidth | 13 GB/s |
| FPS (@512×512) | 76.17 |
| FPS (@720p) | 21.67 |

## 6 Conclusion

In this paper, we design FSS, a novel algorithm, as well as a general NN accelerator for style transfer. First, we present an arbitrary style transfer algorithm named FastStyle, which can achieve higher-quality results with a lower computational cost. In FastStyle, we propose a novel light network to achieve high quality and a new mechanism to avoid retraining when the style changes. Then, we design an NN accelerator for FastStyle through two improvements to the NVDLA architecture. For the first improvement, flexible FSMs are added into NVDLA; then, convolution and Gram operation can be merged into the same datapath, which increases DSP utilization. For the second, the proposed scene change detection and parameter prediction eliminate the off-chip memory traffic of the instance normalization layer, and an 18.6% performance improvement is achieved. The FPGA implementation reaches about 20 FPS when performing style transfer on a 512×512 video stream. The ASIC implementation achieves about 22 FPS on a 720-p video.

**Acknowledgements** This work was supported in part by National Natural Science Foundation of China (Grant No. 62074041), State Key Laboratory of ASIC and System (Grant No. 2021KF009), Zhuhai Fudan Innovation Institute, Key R&D program of Shandong Province (Grant No. 2022CXGC010504).

**References**

1 Wronski B, Garcia-Dorado I, Ernst M, et al. Handheld multi-frame super-resolution. ACM Trans Graph, 2019, 38: 1–18
2 Dong C, Loy C C, He K, et al. Image super-resolution using deep convolutional networks. IEEE Trans Pattern Anal Mach Intell, 2016, 38: 295–307
3 Sajjadi M S M, Vemulapalli R, Brown M. Frame-recurrent video super-resolution. In: Proceedings of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018. 6626–6634
4 Shen W, Bao W, Zhai G, et al. Video frame interpolation and enhancement via pyramid recurrent framework. IEEE Trans Image Process, 2021, 30: 277–292
5 Wei K, Fu Y, Yang J, et al. A physics-based noise formation model for extreme low-light raw denoising. In: Proceedings of 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. 2755–2764
6 Zamir S W, Arora A, Khan S, et al. CycleISP: real image restoration via improved data synthesis. In: Proceedings of 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. 2693–2702
7 Anwar S, Barnes N. Real image denoising with feature attention. In: Proceedings of 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019. 3155–3164
8 Nah S, Son S, Lee K M. Recurrent neural networks with intra-frame iterations for video deblurring. In: Proceedings of 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019. 8094–8103
9 Jin M, Meishvili G, Favaro P. Learning to extract a video sequence from a single motion-blurred image. In: Proceedings of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018. 6334–6342
10 Gatys L A, Ecker A S, Bethge M. Image style transfer using convolutional neural networks. In: Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. 2414–2423
11 Ota K, Dao M S, Mezaris V, et al. Deep learning for mobile multimedia. ACM Trans Multimedia Comput Commun Appl, 2017, 13: 1–22
12 Motamedi M, Gysel P, Ghiasi S. PLACID: a platform for FPGA-based accelerator creation for DCNNs. ACM Trans Multimedia Comput Commun Appl, 2017, 13: 1–21
13 Luan F, Paris S, Shechtman E, et al. Deep photo style transfer. In: Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017. 6997–7005
14 Yang L, Yang L, Zhao M, et al. Controlling stroke size in fast style transfer with recurrent convolutional neural network. Comput Graphics Forum, 2018, 37: 97–107
15 Yao Y, Ren J, Xie X, et al. Attention-aware multi-stroke style transfer. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019. 1467–1475
16 Luan F, Paris S, Shechtman E, et al. Deep painterly harmonization. Comput Graphics Forum, 2018, 37: 95–106
17 Huang X, Belongie S. Arbitrary style transfer in real-time with adaptive instance normalization. In: Proceedings of 2017 IEEE International Conference on Computer Vision (ICCV), 2017. 1510–1519
18 Park D Y, Lee K H. Arbitrary style transfer with style-attentional networks. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019. 5873–5881

19 Li Y, Fang C, Yang J, et al. Universal style transfer via feature transforms. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017. 385–395
20 Wang H, Li Y, Wang Y, et al. Collaborative distillation for ultra-resolution universal style transfer. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. 1857–1866
21 Wang Z, Zhao L, Chen H, et al. Diversified arbitrary style transfer via deep feature perturbation. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. 7786–7795
22 Zhang C, Zhu Y, Zhu S C. MetaStyle: three-way trade-off among speed, flexibility, and quality in neural style transfer. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence and 31st Innovative Applications of Artificial Intelligence Conference and 9th AAAI Symposium on Educational Advances in Artificial Intelligence, 2019. 1254–1261
23 Johnson J, Alahi A, Li F F. Perceptual losses for real-time style transfer and super-resolution. In: Proceedings of European Conference on Computer Vision, 2016. 694–711
24 Cheng M M, Liu X C, Wang J, et al. Structure-preserving neural style transfer. IEEE Trans Image Process, 2020, 29: 909–920
25 Wang Z, Bovik A C, Sheikh H R, et al. Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Process, 2004, 13: 600–612
26 Ulyanov D, Lebedev V, Vedaldi A, et al. Texture networks: feed-forward synthesis of textures and stylized images. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning, 2016. 1349–1357
27 Chen T, Du Z, Sun N, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, 2014
28 Qiu J, Song S, Wang Y, et al. Going deeper with embedded FPGA platform for convolutional neural network. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2016
29 Chen Y H, Krishna T, Emer J S, et al. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J Solid-State Circuits, 2017, 52: 127–138
30 Ma Y, Cao Y, Vrudhula S, et al. Optimizing the convolution operation to accelerate deep neural networks on FPGA. IEEE Trans VLSI Syst, 2018, 26: 1354–1367
31 Li H, Fan X, Jiao L, et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In: Proceedings of 2016 26th International Conference on Field Programmable Logic and Applications (FPL), 2016. 1–9
32 Lu L, Xie J, Huang R, et al. An efficient hardware accelerator for sparse convolutional neural networks on FPGAs. In: Proceedings of 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2019. 17–25
33 Liang Y, Lu L, Xiao Q, et al. Evaluating fast algorithms for convolutional neural networks on FPGAs. IEEE Trans Comput-Aided Des Integr Circ Syst, 2020, 39: 857–870
34 Zhang C, Di W, Sun J, et al. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. In: Proceedings of the 2016 International Symposium on Low Power Electronics and Design, 2019
35 Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: Proceedings of International Conference on Learning Representations, 2015
36 He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. 770–778
37 Ulyanov D, Vedaldi A, Lempitsky V. Instance normalization: the missing ingredient for fast stylization. 2016. ArXiv:1607.08022
38 Dumoulin V, Shlens J, Kudlur M. A learned representation for artistic style. In: Proceedings of International Conference on Learning Representations, 2017
39 Li Y, Fang C, Yang J, et al. Diversified texture synthesis with feed-forward networks. In: Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017. 266–274
40 Zhang H, Dana K. Multi-style generative network for real-time transfer. 2017. ArXiv:1703.06953
41 Sheng L, Lin Z, Shao J, et al. Avatar-Net: multi-scale zero-shot style transfer by feature decoration. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2018. 8242–8250
42 Ma Y, Kim M, Cao Y, et al. End-to-end scalable FPGA accelerator for deep residual networks. In: Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), 2017
43 Russakovsky O, Su J D, Krause J, et al. ImageNet large scale visual recognition challenge. In: Proceedings of International Journal of Computer Vision, 2015. 211–252
44 Cimpoi M, Maji S, Kokkinos I, et al. Describing textures in the wild. In: Proceedings of 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014. 3606–3613
45 Kingma D P, Ba J. Adam: a method for stochastic optimization. In: Proceedings of International Conference on Learning Representations, 2015
46 Abadi M, Agarwal A, Barham P, et al. TensorFlow: large-scale machine learning on heterogeneous distributed systems. 2016. ArXiv:1603.04467
47 Yu Y, Zhao T, Wang M, et al. Uni-OPU: an FPGA-based uniform accelerator for convolutional and transposed convolutional networks. IEEE Trans VLSI Syst, 2020, 28: 1545–1556
48 Xu H, Wang Y, Wang Y, et al. ACG-engine: an inference accelerator for content generative neural networks. In: Proceedings of 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019. 1–7