

Two-Stream FCNs to Balance Content and Style for Style Transfer

Duc Minh Vo*

SOKENDAI (Graduate University for Advanced Studies)
Tokyo, Japan
vmduc@nii.ac.jp

Akihiro Sugimoto

The National Institute of Informatics
Tokyo, Japan
sugimoto@nii.ac.jp

Abstract

Style transfer is to render given image contents in given styles, and it has an important role in both computer vision fundamental research and industrial applications. Following the success of deep learning based approaches, this problem has been re-launched recently, but still remains a difficult task because of trade-off between preserving contents and faithful rendering of styles. Indeed, how well-balanced content and style are is crucial in evaluating the quality of stylized images. In this paper, we propose an end-to-end two-stream Fully Convolutional Networks (FCNs) aiming at balancing the contributions of the content and the style in rendered images. Our proposed network consists of the encoder and decoder parts. The encoder part utilizes a FCN for content and a FCN for style where the two FCNs have feature injections and are independently trained to preserve the semantic content and to learn the faithful style representation in each. The semantic content feature and the style representation feature are then concatenated adaptively and fed into the decoder to generate style-transferred (stylized) images. In order to train our proposed network, we employ a loss network, the pre-trained VGG-16, to compute content loss and style loss, both of which are efficiently used for the feature injection as well as the feature concatenation. Our intensive experiments show that our proposed model generates more balanced stylized images in content and style than state-of-the-art methods. Moreover, our proposed network achieves efficiency in speed.

1. Introduction

How New York looks like in “The Starry Night” by Vincent van Gogh is an interesting question and, at the same time, difficult to answer. In practice, re-painting a famous fine-art style takes much time and requires well-trained artists. Answering this question can be stated as the problem of migrating semantic content of one image to different styles, and it is called style transfer.

Style transfer is long-standing and has fallen into the im-

age synthesis problem which is a fundamental research in computer vision. Style transfer has its origin from non-photo-realistic rendering [7] and is closely related to texture synthesis and color transfer [8, 9]. Along with the impressive progress of various tasks in computer vision using deep neural networks, this topic has recently been re-launched in both academy and industry. [3] showed that the image representation derived from a Convolutional Neural Network (CNN) can be used to represent the semantic content of an image and the style, which opened up a new trend of CNN-based style transfer.

CNN-based approaches in style transfer fall into two categories [10]: Image-Optimisation-Based Online Neural Methods (IOB-NST) and Model-Optimisation-Based Offline Neural Methods (MOB-NST). The key idea of IOB-NST is to synthesis a stylized image by directly updating pixels in the image iteratively through the back-propagation. The IOB-NST such as [3, 11, 12] starts with a noise image and iteratively updates the image by changing the distribution of noise along with the statistics of content and style until the defined loss function is minimized. MOB-NST such as [1, 2, 4, 5, 6, 13, 14, 15, 16, 17], on the other hand, first optimizes a generative model through iterations, and then renders the stylized image using a forward pass. In order to optimize the generative model, MOB-NST trains each feed-forward model for each specific style by using the gradient descent over a large dataset. IOB-NST is known to produce better stylized results in quality than MOB-NST [10], while MOB-NST has more efficiency in speed.

Although existing methods [1, 2, 3, 4, 5, 6, 11, 12, 13, 14, 15, 16, 17] show the capability of rendering image contents in different styles, generated stylized images are not always well balanced in content and style. Such methods take care of either the content or the style, but not both, producing unbalanced stylized images. IOB-NST is good at faithfully rendering the style while it tends to lose the content. MOB-NST, on the other hand, preserves more semantic content than the style. How to keep the balance between the content and the style in style transfer is a crucial issue to improve the quality of stylized images. This is because such

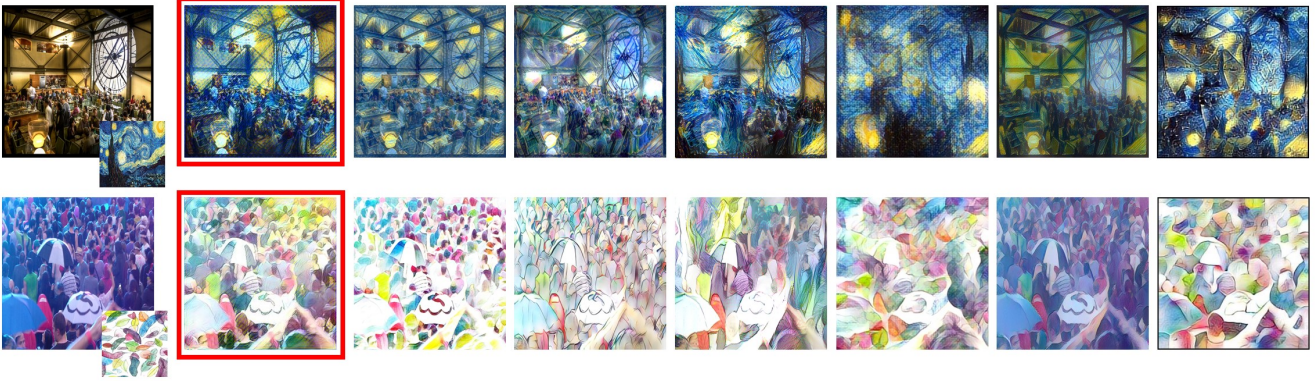


Figure 1: Example of stylized results. Left-most column: content image (large) and style image (small). From left to right: the stylized image by our method, Johnson+ [1], Huang+ [2], and Gatys+ [3], Sheng+ [4], Chen+ [5], and Li+ [6]. Our results surrounded with red rectangles are more balanced in content and style than the others.

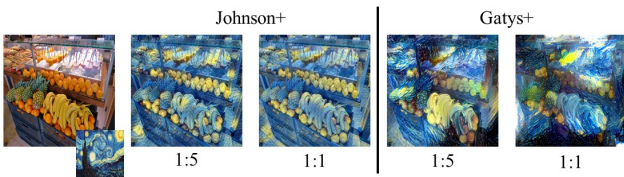


Figure 2: Example of stylized results obtained by Johnson+ [1] and Gatys+ [3] by changing the ratio of content and style from 1:5 to 1:1. Left-most column: content image (large) and style image (small). In each block, from left to right: the stylized image with various ratio of content and style.

balance is required in many applications; for instance, font transfer [18], realistic photo transfer [12, 19]. IOB-NST and MOB-NST have the capability of controlling the balance between the content and the style. Namely, they allow to manually change the ratio of content and style. However, changing the ratio do not guarantee that network parameters for stylized images changes as expected, meaning that the contributions of the content and the style in a stylized image are uncontrollable in reality. Fig. 2 shows examples obtained by IOB-NST (Gatys+ [3]) and MOB-NST (Johnson+ [1]) with various settings of contributions of the content and the style. We can see although the ratio of content and style is significantly changed, the results do not change much.

Another important issue to address is the computational speed. Although MOB-NST such as [1, 2, 4, 5, 6, 13, 14, 15, 16, 17] are able to produce stylized images fast, they rely on a strong computational power. Therefore, either IOB-NST or MOB-NST is hard to apply to real-time applications.

We propose an end-to-end two-stream network for balancing the content and style in stylized images where contributions of the content and the style are adaptively taken

into account. The encoder part of our network consists of the content stream and the style stream where the streams have different architectures. The two streams are connected by adaptive feature injection and independently trained to learn the semantic content or the style representation. The content features and the style features are then combined in our proposed adaptive concatenation to ensure the balanced contribution of each stream. As the decoder part of our network, we use the feed-forward model to reduce the rendering time while we spend much time on learning like [1, 2, 4, 5, 6, 17]. Unlike other methods that train a new model from the scratch for a yet unknown style, we fine-tune parameters from an existing model, allowing our network not only to accommodate fast training but also to easily adapt new styles. Our experiments demonstrate that our method produces more balanced stylized images in both content and style than the state-of-the-art methods (Fig. 1). They also show that our method runs about $22\times$ faster than the state-of-the-art methods. We remark that our proposed model is trained for one style only, but it is easy to be fine-tuned to other styles incrementally with a low cost.

The rest of this paper is organized as follows. We briefly review and analyze related work in Section 2. Next, we analyze the semantic levels of image features for content and style in Section 3. Then, we present the detail of our proposed method in Section 4. Section 5 and Section 6 discuss our experiments. Section 7 draws the conclusion. We remark that this paper extends the work reported in [20]. Our main extensions in this paper are building a new network using both our proposed adaptive feature injection and concatenation, and adding more experiments.

2. Related work

Early work on style transfer was reported in the context of texture synthesis. Some methods there used histogram matching [21] and/or non-parametric sampling [8, 9]. These

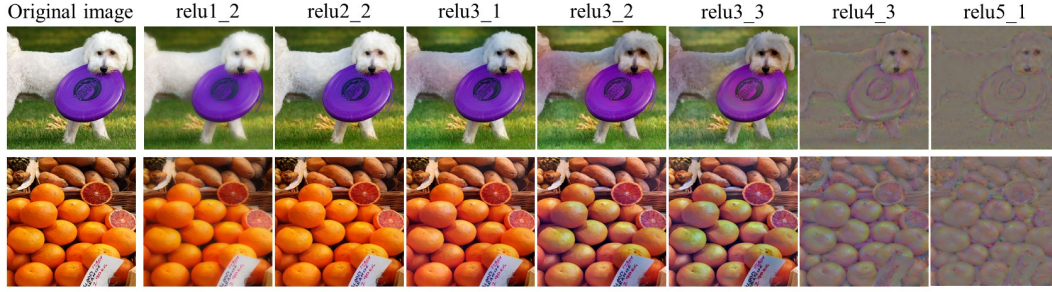


Figure 3: Examples of the feature reconstruction for several layers from the VGG-16 pre-trained network.

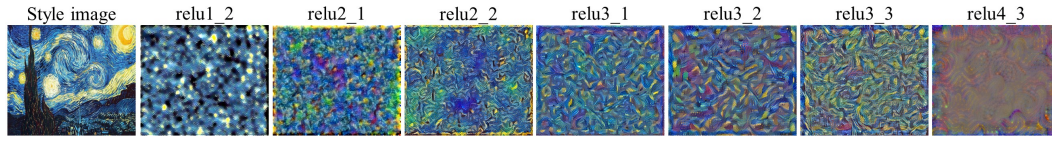


Figure 4: Examples of style image reconstruction for several layers from the VGG-16 pre-trained network.

methods had limited results because they relied on hand-crafted low-level features and often failed in capturing features in semantic levels from the content and the style.

[3] for the first time proposed a method using CNNs and showed remarkable results. Their method trains CNNs to learn the semantic information from content images and matched it with the distribution of the style. It starts from a randomly distributed noise image and iteratively updates the image to produce an image satisfying the semantic distribution of the content image and appearance statistics of the style. During the iteration, the weighted sum of style loss and content loss is minimized. As follow-up work of [3], [11] proposed a structure preservation method using Matting Laplacian for photo-realistic style transfer. [12] utilized the screened Poisson equation to make a stylized image more photo-realistic. [22] proposed a Laplacian loss that computes the Euclidean distance between the Laplacian filters responding to a content image and a stylized image in order to keep a fine structure of the content image. These approaches fall into the IOB-NST category, and all face with the computational speed problem.

[1] and [23], on the other hand, took MOB-NST, proposing a feed-forward CNN and used the perceptual loss function for gradient-based optimization. The perceptual loss used there is similar to content and style loss in [3]. Their models have only to pass the content image to a single forward network to produce a stylized image, which is fast. Their two models are different only in the network architecture. [1] follows the design of [24] with their modification of using residual blocks and fractionally strided convolutions while [23] uses a multi-scale in their generator. [17] also utilized the feed-forward network, and they used multiple-generator to improve the quality of results. These methods are fast in generating stylized images, but they are capable

of dealing with a single style only.

[25] proposed a multi-style network that introduces shared-computation in many style images where they used instance normalization (IN) [26] for balancing features from the content and from the style. They also proposed an improvement of IN to learn a different set of affine parameters for multi-styles in the batch way. However, their model can train a limited number of styles because the network capability is limited, meaning that the number of styles to handle is limited. [5] proposed a method that overcomes the limitation of the number of styles by using a patch-based method. Their method first extracts a set of patches from the content and style each, and then, for each content patch, the method finds its closest style patch and swaps their activation. In this way, their method transfers an unlimited number of styles; however, the cost for patch extraction and swapping increases the computational time significantly. [6] also proposed a method for multi-style transfer using feature transformations. They first employ pre-trained VGG-19 as their encoder to train an decoder for image reconstruction. Then, with fixing both encoder (VGG-19) and decoder, their model performs the style transfer through whitening and coloring transforms on a given content image and a style image. Though their method successfully solves the multi-style transfer, it still suffers from the computational cost and loses the content due to the feature transformations.

[2] and [4] proposed multi-style transfer models consisting of two CNN streams for content and style. [2] employed the pre-trained VGG-16 to extract content and style features and introduced Adaptive Instance Normalization (AIN) to make the mean and the variance of content features similar to those of style features. [4], on the other hand, proposed AvatarNet which employed the pre-trained VGG-19

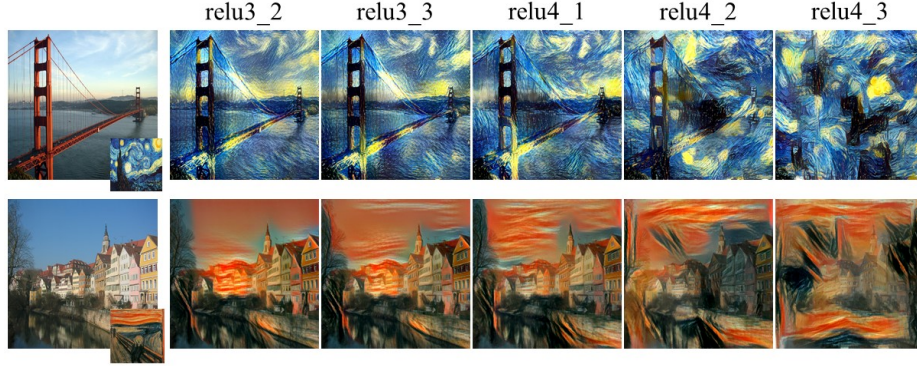


Figure 5: Examples of combination of content and style images from *relu3_2* to *relu4_3*. Left-most column: content image (large) and style image (small), From left to right: the stylized images at different combination levels by Gatys+ [3] where the ratio of contributions of content and style is 1:1.

to extract the content and style features. These features are matched by using style-swap [5] or AIN [2] before being fed into the decoder. Different from [2], their models have skip-connections from the style encoder to the decoder. [2] and [4], however, used the same architecture for the content CNN and for the style CNN. Having the same CNN architecture for the content and the style causes unavoidable unbalance between the content and the style because semantic levels extracted from the content and the style should not be the same in style transfer. Those models require expensive computational cost as well. Furthermore, AIN [2] assumes the standard distribution on pixel values of images, which is not always ensured in styles when normalizing data. In deed, AIN [2] tends to produce a lot of artifacts; especially they are visible on flat surfaces [16]. We remark that the skip-connection in AvatarNet [4] weights the style contribution more, causing unbalance in stylized images.

Along with using Generative Adversarial Network (GAN) [27] in image synthesis, several GAN-based models for style transfer are also proposed [13, 14, 16, 28]. These models also optimize the network with a large number of content images during the training step, and thus fall in the MOB-NST category. Though GAN-based models bring a promising approach to improve the quality of stylized images, their results, at this time, still are less impressive [10]. Furthermore, as in common with other GAN-based approaches, their training processes are also unstable.

Different from the methods above, we take into account the contributions of the content and the style through a two-stream feed-forward network to balance the content and the style in stylized images. In particular, our proposed two-stream network is different from [2, 4] in that our network has different depths in layer for the content and the style encoders to extract different semantic levels of the content and the style. In addition, separating content and style enables our method easy to fine-tune to other styles with a cheaper

computational cost (re-training time, required numbers of training images) than other models possessing only one encoder [1, 5, 17]. As a result, our method is able to easily deal with multi-styles.

3. Semantic levels of image features for content and style

Along with the depth, CNN is known to extract different semantic levels of image features in layers. As demonstrated in [1, 3], features in early layers reflect colors, textures, and common patterns of images while those in latter layers preserve content and spatial structure of images. We, therefore, expect that the features in lower layers work as style features and those in higher layers do as content features. Using appropriate semantic levels of image features in style transfer is crucial. We thus experimentally exploit the semantic levels of image features in VGG-16 [29] to design suitable numbers of layers in designing our network to extract content and style features. We remark that we refer [1, 3] in which image reconstruction is learned using hidden features in CNN layers.

For the content image reconstruction, we randomly prepare 100 images. We then feed each of the 100 images into the VGG-16 [29] pre-trained on object recognition using ImageNet dataset [30] without any fine-tuning and extract the features at each Rectified Linear Unit (ReLU) [31]. These features are employed to reconstruct original images using inverting technique [32]. Hereafter, we use *reluX_Y* to mention a specific ReLU layer; see the definition of VGG-16 [29] architecture for details. Fig. 3 shows some examples of image reconstruction at several layers. We see that at low levels, i.e., from the 2nd layer (*relu1_2*) to the 5th layer (*relu3_1*), the reconstructed images are similar to the original image, meaning that these layers successfully keep colors, textures, and common patterns of images. At higher levels, i.e., from the 6th layer (*relu3_2*) to the 10th

layer (*relu4.3*), the reconstructed images preserve the content and spatial structure. At even higher layers that start from the 11th layer (*relu5.1*), semantic features are gradually learned; the exact shape, on the other hand, is not preserved.

For the style image reconstruction, we use Adam optimization [33] to find an image that minimizes the style reconstruction loss (proposed in [3]). To obtain style reconstructed images, we start from a noise image and optimize the style loss as [3] using the VGG-16 pre-trained on ImageNet. Fig. 4 shows an example of the style image reconstruction. We see that the style of image can be obtained until the 7th layer (*relu3.3*)

The above observation holds true for the images and the styles that we evaluated. Combining the insight given by [1, 3], we may thus conclude that the low-level layers reflect the style of the image while the high-level layers capture the content of the image. More precisely, from the 6th layer (*relu3.2*) to the 10th layer (*relu4.3*), the network is capable of appropriately capturing content information in the images. The style information, on the other hand, can be obtained from the 2nd (*relu1.2*) to the 7th (*relu3.3*) layers.

[3] pointed out that image content and style cannot be completely disentangled. This indicates that depending on the objective, we have to appropriately design the layer levels of content and style features for their combination. We thus further analyze effectiveness of the layers from the 6th (*relu3.2*) to the 10th (*relu4.3*) for content matching to determine the best one for combination. We follow [3] to synthesize the stylized images where we set the contributions of content and style to be equal with each other. To this end, we fix the style matching from the 2nd (*relu1.2*) to the 7th (*relu3.3*) layers, while performing the content matching at every single layer from the 6th (*relu3.2*) to the 10th layers (*relu4.3*). Fig. 5 shows examples of stylized images having different layers in combination. We see that the content matching at the 6th and the 7th layers (*relu3.2* and *relu3.3*) is most reasonable to keep the balance of content and style in stylized images.

Using above observation, we design our network to fully exploit the characteristics of image features. We choose the 6th layer for content because it has a smaller number of parameters than the 7th layer (it is faster to learn). We choose the 4th layer for style because it is neither too early in layer nor marginally different from the layer used for content. In conclusion, we use the features at the 6th layer (*relu3.2*) for content and those at the 4th layer (*relu2.2*) for style.

4. Proposed method

4.1. Network design

Our network follows end-to-end encoder-decoder architecture for rendering of the content in a given style [1, 5, 17].

The network in [1, 5, 17] possesses only one encoder to extract the semantic content and style. This means that the extracted semantic level of the content and that of the style are the same. When we stylize images, the role of the content should be different from that of the style because the content gives us what exist (object shapes and locations) in the rendered image and the style gives us the impression of the rendered image. Accordingly, the semantic level used for the rendering should be different depending on the content or the style. Otherwise, unbalance between the content and style remains in stylized images. We thus design a network having two encoders in which their architectures are different from each other to extract different semantic levels of the content and the style. With the two encoders, our model treats the content and the style in different ways, allowing the network to be able to balance the roles of the content and the style better than the model having only one encoder.

Ideally, the network should be able to retain the semantics of the content as well as the statistics of the style as much as possible. The semantic content and style of an image are captured at different layers in the network (see [1, 3] and Section 3): the network obtains the style at low-level layers in depth while high-level layers become more sensitive to the actual content of the image. We thus design the encoders with different depths to retain useful information from both the content and the style. Namely, we design a deep encoder for the content and a shallow encoder for the style. Moreover, in order to reflect features extracted from the style at low-level to those from the content, we employ the feature injection via the skip-connection technique from the shallow encoder to the deep one. Because the content feature and the style feature are extracted at different levels in the network, they have different characteristics. We thus introduce an effective concatenation to enhance the contribution of these features for good performances instead of implementing their simple ones.

4.2. Network architecture

Our proposed network consists of three Fully Convolutional Network (FCNs): two encoders and one decoder (Fig. 6). The two encoders are a deep network, the content subnet, to extract content feature ϕ_c from a content image, and a shallow network, the style subnet, to extract style feature ϕ_s from a style image. The feature injection is employed between the content subnet and the style subnet using the balance weight (cf. Section 4.4). This balance weight is also used to adaptively concatenate the features ϕ_c and ϕ_s at the top of content and style subnet before being fed into a deep network, the generator subnet, to produce a stylized image. We employ the VGG-16 model [29] as the loss network in the training phase.

Our network receives the content and style images where each image is with the size of $n \times n \times 3$ (n is the size of

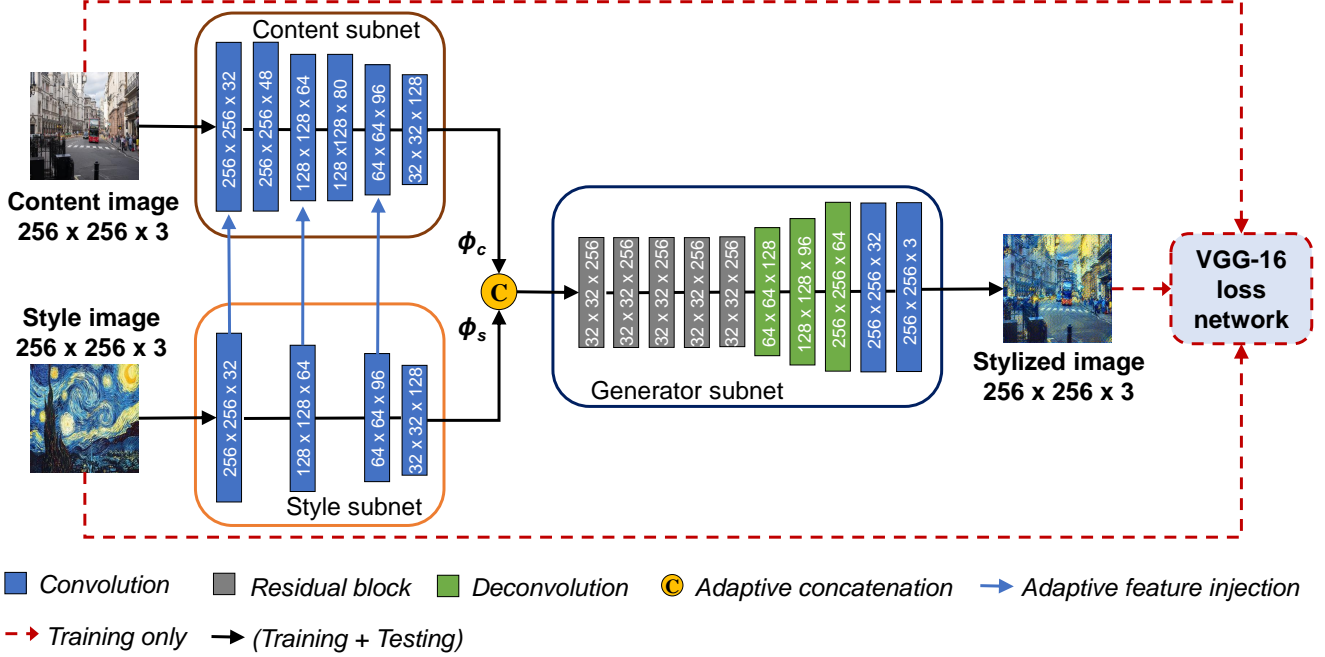


Figure 6: Framework of our proposed method. Our network consists of two encoders having different architectures and one decoder. The loss network is used to train the encoders and the decoder.

image, 3 are for RGB channels), and synthesizes an stylized image of $n \times n \times 3$. In the training phase, we use the images of $256 \times 256 \times 3$ ($n = 256$). Although we train the network on images with the size of $256 \times 256 \times 3$, the network can accept any size of images in testing (n can be 64, 128, 256, or 512). We remark that the size of the content image and that of the style image have to be the same to ensure the consistency of the feature size when injecting and concatenating the content and the style features.

4.2.1 Content subnet

The content subnet is a stack of six convolution layers with the filter size of 3×3 , and the padding size of 1×1 . We use the stride of 2×2 at the third, the fifth, and the sixth layers to reduce the size of feature maps and the stride of 1×1 at the other layers. The numbers of the output channels are 32, 48, 64, 80, 96, and 128, respectively. Each convolution layer is followed by a spatial instance normalization (IN) layer [26] and a Rectified Linear Unit (ReLU) layer [31]. In order to avoid the border artifacts caused by convolution, the reflection-padding is used instead of the zero-padding similarly to [25].

4.2.2 Style subnet

The style subnet, which has four convolution layers, is shallow network (more precisely, shallower than the content

subnet). All convolution layers have the filter size of 3×3 , the reflection-padding of 1×1 , and the stride of 2×2 , except for the first layer that employs the stride of 1×1 . The numbers of the output channels are 32, 64, 96, and 128, respectively. Similarly to the content subnet, each convolution layer is also followed by an IN layer [26] and a ReLU layer [31].

We employ feature injection from the feature ϕ_s^q at the q -th layer in the style subnet to those ϕ_c^p at the p -th layer in the content subnet, the size of whose feature map is the same (Table 1). To take into account the contributions of ϕ_s^q and ϕ_c^p , we introduce the adaptive feature injection with the balance weight (cf. Section 4.4).

4.2.3 Generator subnet

The generator subnet consists of five residual blocks, three deconvolution layers, and two convolution layers in this order.

[1] argues that the residual block can enrich the information involved in the input feature. We, therefore, use residual blocks to increase the impact of the balance weight in the concatenated feature. Similarly to [1], we use five residual blocks outputting 256 channels, where each of them has two convolution layers with the filter size of 3×3 , the reflection-padding of 1×1 , the stride of 1×1 , and a summation layer as in [34]. All convolution layers are followed by an IN layer [26] (we use it to replace the batch normal-

Table 1: Architecture of our encoders. The arrow (\leftarrow) indicates the adaptive feature injection.

Content subnet			Style subnet		
No	Layer	Output channel	No	Layer	Output channel
0	Content image	3	0	Style image	3
1	Convolution	32	1	Convolution	32
2	Instance normalization	32	2	Instance normalization	32
3	ReLU	32	\leftarrow 3	ReLU	32
4	Convolution	48			
5	Instance normalization	48			
6	ReLU	48			
7	Convolution	64	4	Convolution	64
8	Instance normalization	64	5	Instance normalization	64
9	ReLU	64	\leftarrow 6	ReLU	64
10	Convolution	80			
11	Instance normalization	80			
12	ReLU	80			
13	Convolution	96	7	Convolution	96
14	Instance normalization	96	8	Instance normalization	96
15	ReLU	96	\leftarrow 9	ReLU	96
16	Convolution	128	10	Convolution	128
17	Instance normalization	128	11	Instance normalization	128
18	ReLU	128	12	ReLU	128

ization [35] in the original architecture [34]) and a ReLU layer [31].

To upscale the feature map, we employ three deconvolution layers with the same filter size of 3×3 , the reflection-padding of 1×1 , and the stride of 2×2 , outputting 128, 96, and 64 channels, respectively.

In order to eliminate the affect of the convolution stride, we use two convolution layers which have the filter size of 1×1 , the padding of 0×0 , and the stride of 1×1 , outputting 32 and 3 channels. All deconvolution layers and convolution layers are followed by an IN layer [26] and a ReLU layer [31], except for the last convolution layer that uses the tanh activation to guarantee that the range of the output can be normalized to be $[0, 255]$.

4.3. Loss function

We employ two loss functions for content loss and style loss, which are computed from layers of the loss network. The content loss \mathcal{L}_c computes the similarity of high-level features between the content image and the stylized image. The style loss \mathcal{L}_s , on the other hand, computes the similarity of low-level features between the style image and the stylized image.

The overall loss is a weighted sum of the content loss and the style loss:

$$\mathcal{L}(\hat{y}, y_c, y_s) = \alpha \mathcal{L}_c(\hat{y}, y_c) + (1 - \alpha) \mathcal{L}_s(\hat{y}, y_s), \quad (1)$$

where y_c, y_s , and \hat{y} denote the content image, the style, and the stylized image, respectively. α is the combination weight (we set $\alpha = 0.5$ in our experiments to equally weight these two loss functions).

We obtain the content loss at M layers as follows:

$$\mathcal{L}_c(\hat{y}, y_c) = \frac{1}{M} \sum_{k \in M} \frac{1}{C_k \times H_k \times W_k} \|\Phi_k(\hat{y}) - \Phi_k(y_c)\|_2, \quad (2)$$

where $\Phi_k(\cdot)$ denotes the normalized feature map at the k -th layer, which has $C_k \times H_k \times W_k$ elements. The range of \mathcal{L}_c is $[0, 1]$.

The style loss is computed at N layers as follows:

$$\mathcal{L}_s(\hat{y}, y_s) = \frac{1}{N} \sum_{k \in N} \|G(\Phi_k(\hat{y})) - G(\Phi_k(y_s))\|_F, \quad (3)$$

where $\|\cdot\|_F$ denotes the Frobenius norm [36]. $G(\Phi_k(\cdot))$ is the Gram matrix [36] of the normalized feature map at the k -th layer. The Gram matrix $G_{C_k \times C_k}$ has elements $G_{ij} = \langle v_i, v_j \rangle$ where v_i, v_j are features at the i -th and the j -th channels respectively of the feature map $\Phi_k(\cdot)$. The range of \mathcal{L}_s is $[0, 1]$.

4.4. Adaptive feature injection and concatenation

In our network, we employ the feature injection between the content features and the style features. We also concatenate them to feed into the generator subnet. To weight the contributions of the content features and the style features, we introduce the balance weight γ . This balance weight is adaptively updated during the training so that it retains the balance between the content and the style in stylized images.

At the t -th iteration in training phase, γ_t is computed as follows:

$$\gamma_t = \frac{\mathcal{L}_s(t)}{\mathcal{L}_s(t) + \mathcal{L}_c(t)}, \quad (4)$$

where $\mathcal{L}_s(t)$ and $\mathcal{L}_c(t)$ are the style loss and the content loss at the t -th iteration in the training phase. To restrict the fluctuation of the balance weight, we compute γ at every non-overlapping T iterations and use it for the next T iterations:

$$\gamma = \frac{1}{T} \sum_{t=1}^T \gamma_t. \quad (5)$$

Using γ , we sum up the content feature at the p -th layer ϕ_c^p and the style feature at the q -th layer ϕ_s^q for the feature in adaptive feature injection as follows:

$$\phi^{pq} = (\gamma \times \phi_c^p) + ((1 - \gamma) \times \phi_s^q). \quad (6)$$

Similarly, we concatenate the content feature ϕ_c and the style feature ϕ_s in the adaptive concatenation as follows:

$$\phi = (\gamma \times \phi_c) \oplus ((1 - \gamma) \times \phi_s). \quad (7)$$

The learned balance weight γ ensures the balance of the contributions of the content feature and the style feature in both feature injection and concatenation layers. For example, when \mathcal{L}_s is smaller than \mathcal{L}_c (meaning $\gamma \leq 0.5$ in Eq. (5)), the contribution of style feature is increased in the next iterations, and vice versa. Moreover, the learned balance weight γ is more advantageous than the fixed balance weight that does not concern the balance of losses.

In order to explicitly control the contribution ratio of the content and the style, we manually set the expected contribution ratio in the loss function, and then introduce the learnable weight that allows us to change stylized images as we expect. The combination weight α takes the former role while the learnable balance weight γ does the latter role. In other words, in our method, α sets an expected contribution ratio of content and style in stylized images through the loss function while γ controls the learning direction of the network during the training to achieve the contribution ratio specified by α . In our experiments where we set $\alpha = 0.5$, we see that γ works for the equal contribution ratio of the content and the style as expected (see Sections 6.1 and 6.2 for details). We remark that α and γ together play the role of the indicator for how much the content and the style are emphasized in obtained stylized images.

5. Experimental setup

5.1. Dataset and compared methods

5.1.1 Dataset

We used in our experiments, images in the MS-COCO 2014 dataset [37] as our content images, and six famous paintings widely used in style transfer [1, 2, 3], as our style images (cf. Fig. 7).

We used the MS-COCO 2014 training set for our training, and we randomly selected 20 images from the MS-COCO 2014 validation set for our validation. In the testing phase, on the other hand, we randomly selected 50 images from MS-COCO 2014 validation (different ones from the 20 images used in our validation).

5.1.2 Compared methods

We compared our method with SOTA methods: Gatys+ [3], Johnson+ [1], Huang+ [2], Sheng+ [4], Chen+ [5], and Li+ [6]. We note that Gatys+ is based on IOB-NST and the others are on MOB-NST. For Gatys+, we used the re-implementation version by J. Johnson¹. For the others, we used publicly available source codes with parameters recommended by the authors (Johnson+², Huang+³, Sheng+⁴, Chen+⁵, Li+⁶). We remark that we set 1000 iterations for Gatys+.

¹<https://github.com/jcjohnson/neural-style>

²<https://github.com/jcjohnson/fast-neural-style>

³<https://github.com/xunhuang1995/AdaIN-style>

⁴<https://github.com/LucasSheng/avatar-net>

⁵<https://github.com/rtqichen/style-swap>

⁶<https://github.com/Yijunmaverick/UniversalStyleTransfer>

5.2. Implementation details

5.2.1 Implementation setup

We implemented our method in PyTorch⁷. We used the instance incremental learning strategy for dealing with multiple styles. We conducted all experiments using a PC with CPU core i7 3.7 GHz, 12 GB of RAM, and GTX 770 GPU (4 GB of VRAM).

We performed the adaptive feature injection from layers $q = 3, 6, 9$ in the style subnet to layers $p = 3, 9, 15$ in the content subnet, respectively (Table 1). We adopted the VGG-16 model [29] pre-trained on the ImageNet [30] as the loss network without any fine-tuning. All layers after *relu4_3* layer were dropped. We obtained the content loss at $M = 1$ layer, e.g., *relu4_3*, and the style loss at $N = 3$ layers, e.g., *relu1_2*, *relu2_2*, and *relu3_3* (M and N are defined in Section 4.3).

5.2.2 Training the model

Our method addresses a one-style model to reduce computational time. For training a new yet unknown style, we fine-tune parameters from an existing model. With this learning strategy, our method can easily adapt a new style with a lower cost than existing work [1, 2, 3, 17]. Moreover, the fine-tuning learning enables our method to deal with an unlimited number of styles fast unlike existing methods such as [5, 25].

We first trained an initial model on the Starry Night style and then incrementally fine-tuned on the other styles one by one. We trained the network on the Starry Night style with a batch size of 2 for 80k iterations corresponding to 2 epochs. The balance weight γ in Eq. (5) is re-computed at every $T = 500$ iterations. All the training and validation images are resized to 256×256 . To train the model, we used the Adam optimizer [33] with the learning rate of 10^{-3} , the moments $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and the division from zero parameter $\epsilon = 10^{-8}$. We did not use the learning rate decay and the weight decay.

For the initial model, we trained all subnets simultaneously with independently updating the weight of each subnet. Validation was performed at every 100 iterations during the training process. When observing the content loss and the style loss on the validation set, if any loss function raises the overfitting problem, we stopped updating the weight of the corresponding subnet.

We incrementally fine-tuned the initial model to the other styles one by one. 2000 images in the MS-COCO 2014 training set [37] were randomly selected as content images for training. The network was trained for 1000 iterations with the batch size of 2. The Adam optimizer [33] was also used with the same parameters as the training of the initial model. The balance weight γ in Eq. (5) was re-computed at

⁷<https://pytorch.org/>

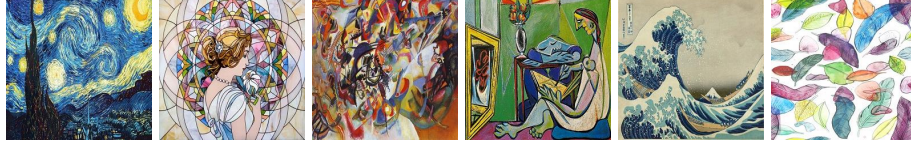


Figure 7: Styles used in experiments. From left to right: Starry Night, Mosaic, Composition VII, La Muse, The Wave, and Feathers.

every $T = 50$ iterations. The loss-based training technique was also applied to avoid overfitting, where the validation was performed at every 50 iterations.

5.3. Evaluation metric

In order to evaluate the quality of synthesized images, most previous work employed user studies although they are subjective and have ambiguity in evaluation. We, on the other hand, evaluate stylized images by quantifying the content and style losses. Intuitively, when the total loss is sufficiently small, we may say that the overall quality of stylized images is good. Furthermore, the quality of stylized images also depends on how the content and the style are reflected in them. We have to consider these two factors in evaluating the quality of stylized images. Since the contributions of the content and the style are controlled by α (set in advance) in our method, we may see if a synthesized image is good in quality by evaluating (i) whether its total loss is sufficiently small, and (ii) whether the ratio between its content and style losses consistently agrees with the preset contribution ratio (i.e., combination weight α) between the content and the style. We thus introduce a metric to evaluate the quality of synthesized images using these two criteria. We remind that we set $\alpha = 0.5$ (for simplicity) in our experiments to see the content and style losses converge to almost the same values.

For each pair of content image c and style image s , we compute content loss \mathcal{L}_c and style loss \mathcal{L}_s . In the 2D plane whose coordinate system is defined by content loss and style loss, the criterion (i) can be measured using the distance between the origin and $(\mathcal{L}_c, \mathcal{L}_s)$. The criterion (ii), on the other hand, can be measured by evaluating how close $(\mathcal{L}_c, \mathcal{L}_s)$ is to the line of “content loss”=“style loss” (called the balanced axis hereafter).

We assume that we have K stylized images. We normalize content loss and style loss for each stylized image over K images:

$$\widetilde{\mathcal{L}}_c = \frac{1}{1 + \exp\left(\frac{\mathcal{L}_c - \overline{\mathcal{L}}_c}{\sigma_c}\right)}, \quad \widetilde{\mathcal{L}}_s = \frac{1}{1 + \exp\left(\frac{\mathcal{L}_s - \overline{\mathcal{L}}_s}{\sigma_s}\right)}, \quad (8)$$

where $\overline{\mathcal{L}}_c$, σ_c , $\overline{\mathcal{L}}_s$, and σ_s are the mean and the standard deviation of content loss and style loss over K stylized images, respectively.

The quality of stylized images with respect to the criterion (i) is measured using

$$length = \sqrt{\widetilde{\mathcal{L}}_c^2 + \widetilde{\mathcal{L}}_s^2}. \quad (9)$$

Let $\omega \in [0, \frac{\pi}{4}]$ denote the angle between the line going through the origin and $(\widetilde{\mathcal{L}}_c, \widetilde{\mathcal{L}}_s)$ and the content loss axis or the style loss axis (the smaller angle is selected):

$$\omega = \begin{cases} \tan^{-1} \frac{\widetilde{\mathcal{L}}_s}{\widetilde{\mathcal{L}}_c} & \text{if } \widetilde{\mathcal{L}}_c \geq \widetilde{\mathcal{L}}_s \\ \pi/2 - \tan^{-1} \frac{\widetilde{\mathcal{L}}_s}{\widetilde{\mathcal{L}}_c} & \text{otherwise} \end{cases}. \quad (10)$$

Larger ω indicates that $(\widetilde{\mathcal{L}}_c, \widetilde{\mathcal{L}}_s)$ is closer to the balanced axis, meaning that the stylized image is more balanced in content and style. This reflects the criterion (ii).

Using $length$ and ω above, we define our metric *balance*:

$$balance = \frac{\tan(\omega)}{length}. \quad (11)$$

balance concerns both the two criteria (i) and (ii). Therefore it is a useful metric for evaluating stylized images. We note that larger *balance* is better because $\tan(\omega)$ should be larger and *length* should be smaller for better stylized images.

6. Experimental results

6.1. Qualitative evaluation

Figure 8 shows examples of the obtained results, showing that the stylized images obtained by our method are more balanced in content and style. We also see that overall the results obtained by Gatys+ [3], Sheng+ [4], and Li+ [6] reflect the style well, but they mostly lose content (we cannot understand the content of stylized results using La Muse and Feathers styles). In some styles (Starry Night, Composition VII, and The Wave), we see that Johnson+ [1] seems to randomly select a patch in the style and paste it into the content image. Huang+ [2] also loses the content and suffers from a so-called checkerboard effect. We also see that Chen+ [5] loses almost style and tends to keep the original content images.

To objectively compare the obtained results, we conducted three user studies, including overall quality, content

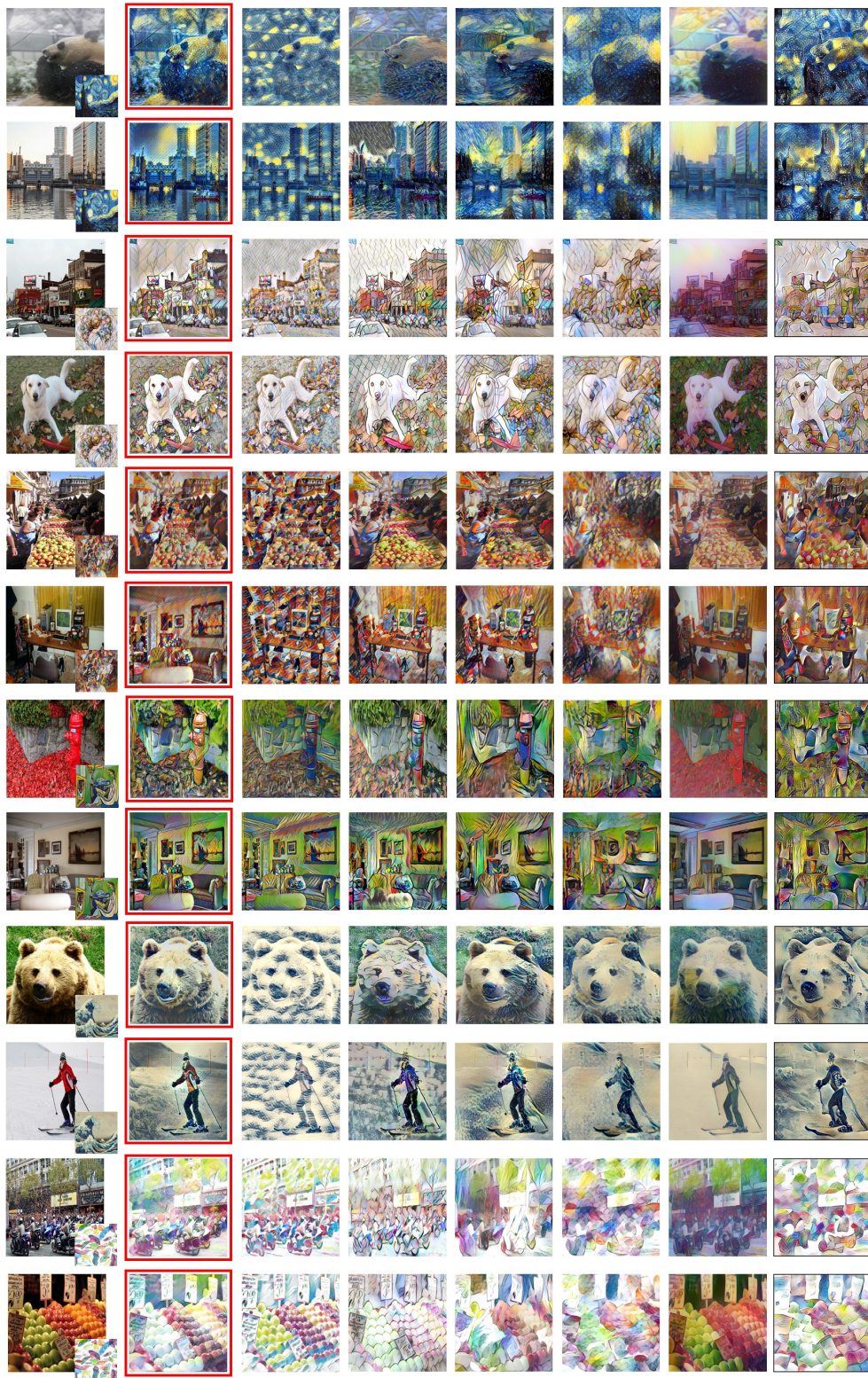


Figure 8: Visual comparison of our method against the state-of-the-art methods. Left-most column: content image (large) and style image (small). From left to right: the stylized image by our method, Johnson+ [1], Huang+ [2], and Gatys+ [3], Sheng+ [4], Chen+ [5], and Li+ [6]. Our results surrounded with red rectangles are more balanced in content and style than the others. Note that all stylized images are with the size of 512×512 .

preserving and style look-like. From the visual comparison in Fig. 8, we see that evaluating all stylized results among compared methods is pretty difficult. We thus picked up three methods only for our user studies. To this end, we investigated the quantitative comparison (Section 6.2). As Gatys+ [3] is known to keep styles most while Johnson+ [1] retain the content most, these methods are appropriate to choose for our user studies. Among the remaining compared methods, we see that Huang+ [2] is most balanced (the loss distributions of Huang+ [2] appear near balanced axis (Fig. 10)). We, therefore, chose Gatys+ [3], Johnson+ [1] and Huang+ [2] for our user studies.

For our user studies, we randomly selected 20 images from the 50 testing images as content images and chose 5 styles by excluding The Wave style because it is simpler than the other styles (Fig. 7). We remark that the combination of 20 content images and 5 styles results in 100 stylized images by each method. In each user study, we presented 100 sets of images to 31 subjects where each set consists of a content image, a style image, and four output images obtained by our method and the three comparison methods [1, 2, 3]. We then asked the subjects to rank the four output stylized images at each set (1st is best, and 4th is worst). For the overall quality study, the subjects were asked to give the ranking based on the overall quality at each set. For the content preserving study, the subjects were asked to rank output images in each set based on how faithfully the images preserve the content in content images. For the style look-like study, on the other hand, the subjects ranked output images in each set based on how the images look like the style in style images. We note that four output images are aligned in the random order in each set and that each set was displayed for 6 seconds.

Table 2, Table 3, and Table 4 show the average of rankings over the 100 sets for the overall quality, the content preserving, and the style look-like studies, respectively. We also computed the average of rankings in each style, which is also illustrated in Table 2, Table 3, and Table 4.

We see that our method takes the best ranking among the four methods in overall quality (Table 2). Looking into the results in more detail, we see that our method is ranked in the first place at the Mosaic style, and in the second place at others (except for Composition VII style). This indicates that our method performs stably well in overall quality in accordance with human cognition. We remark that the Composition VII style is rather complex (Fig. 7) and, the results for this style are difficult to evaluate. We also remark that the single-style models (ours, Gatys+ [3], and Johnson+ [1]) performed better than the multi-style model (Huang+ [2]).

For the content preserving (Table 3) and the style look-like (Table 4) studies, our method takes the second best ranking. Note that the scores in these studies more largely distributed than those in the overall quality study. As MOB-

Table 2: Average of rankings in the overall quality study. The best and the second best results are given in **red** and **blue**, respectively.

Style	Ours	Johnson+ [1]	Huang+ [2]	Gatys+ [3]
Starry Night	2.12	2.72	3.14	2.01
Mosaic	2.21	2.25	2.91	2.63
Composition VII	2.47	2.95	2.4	2.18
La Muse	2.38	2.28	2.82	2.51
Feathers	2.15	1.82	3.28	2.74
All together	2.27	2.40	2.91	2.41

Table 3: Average of rankings in the content preserving study. The best and the second best results are given in **red** and **blue**, respectively.

Style	Ours	Johnson+ [1]	Huang+ [2]	Gatys+ [3]
Starry Night	2.53	1.96	2.67	2.84
Mosaic	2.13	1.60	3.05	3.22
Composition VII	3.02	1.81	2.50	2.67
La Muse	1.99	1.82	3.06	3.13
Feathers	2.02	1.81	2.50	2.67
All together	2.34	1.80	2.87	2.99

NST is known to perform better in content preserving than IOB-NST [10]; Johnson+ [1], which is MOB-NST, takes the best ranking in the content preserving study. Gatys+ [3], on the other hand, which is IOB-NST, takes the best ranking in the style look-like study. In contrast, our method is ranked in the second place for all styles in the content preserving study (except for Composition VII style) (Table 3) and in the style look-like study (except for Feathers style) (Table 4). These indicate that our method stably produces stylized images balanced in content and style for almost all the styles. We remark that in the case of the Feathers style, the two best methods for the look-like study follow the MOB-NST approach. As MOB-NST is known not to keep styles well [10], this suggests that the Feathers style is a difficult style for users to evaluate stylized images.

6.2. Quantitative evaluation

In order to quantitatively evaluate the obtained results, we computed the averages of *length*'s and *balance*'s over 300 (= 50 contents \times 6 styles) sets for each method (Table 5). We see that our method performs best both in *length* and *balance*. We also computed the averages of *length*'s and *balance*'s in each style, which is illustrated in Fig. 9. Fig. 9 shows that our method performs best in *length* and best in *balance* for all the styles.

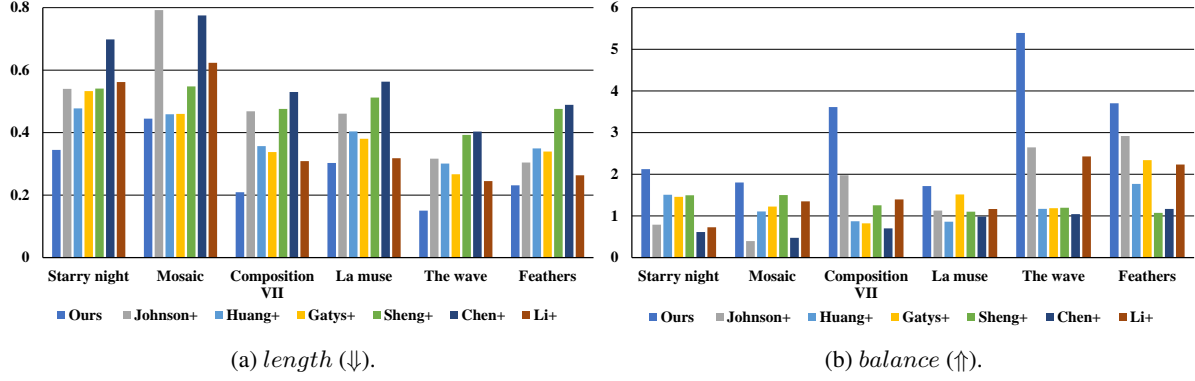


Figure 9: Averages of *length* and *balance* in each style.

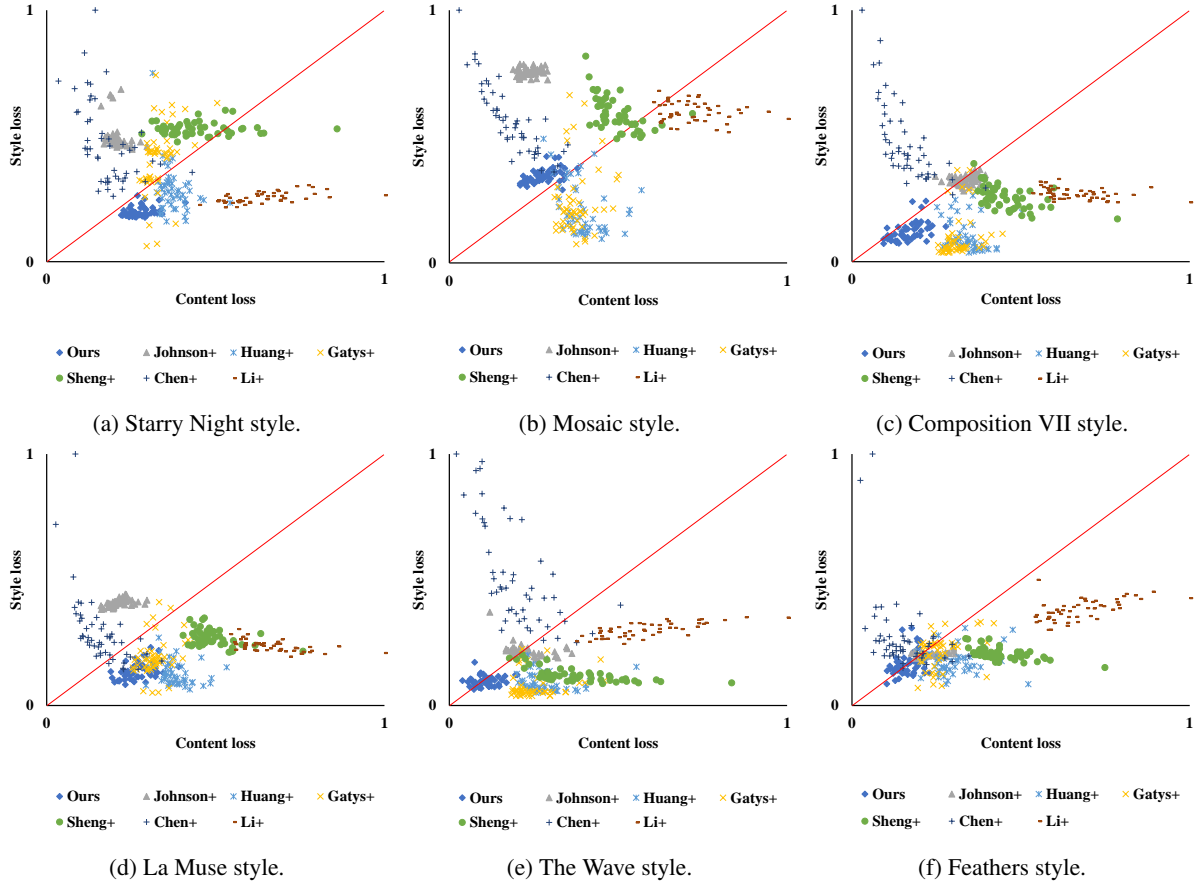


Figure 10: Loss distribution in each style. Red lines denote the balanced axis. Our method has the distributions nearer the balanced axis than the other methods.

To look into the results in more detail, we show the loss distribution of 50 stylized images in each style (Fig. 10). We see that (1) the content loss and the style loss (for each stylized result) in our method are similar with each other and that (2) loss distributions in our method appear densely near the balanced axis for all the styles while those in the other methods do not.

6.3. Computational speed

We measured the running time for generating 300 stylized images with the sizes of 256×256 and 512×512 by each method and compared the average for generating one stylized image by each method.

Table 6 illustrates the average of the running time in

Table 4: Average of rankings in the style look-like study. The best and the second best results are given in **red** and **blue**, respectively.

Style	Ours	Johnson+ [1]	Huang+ [2]	Gatys+ [3]
Starry Night	2.27	2.77	3.34	1.61
Mosaic	2.26	2.66	2.94	2.13
Composition VII	2.49	2.96	2.65	1.90
La Muse	2.71	2.81	2.81	1.67
Feathers	1.69	2.34	3.42	2.55
All together	2.28	2.71	3.03	1.97

Table 5: Averages of *length* (smaller is better) and *balance* (larger is better).

Method	<i>length</i> (\Downarrow)	<i>balance</i> (\Uparrow)
Ours	0.37	2.95
Johnson+ [1]	0.54	1.60
Huang+ [2]	0.45	1.23
Gatys+ [3]	0.45	1.36
Sheng+ [4]	0.52	1.21
Chen+ [5]	0.59	0.72
Li+ [6]	0.49	1.40

Table 6: The average wall-clock time in second for producing one stylized image.

Method	Image size		Implemented framework
	256 × 256	512 × 512	
Ours	0.05	0.18	PyTorch
Johnson+ [1]	1.12	3.79	Torch
Huang+ [2]	1.98	6.78	Torch
Gatys+ [3]	74.12	269.74	Torch
Sheng+ [4]	3.04	10.67	TensorFlow
Chen+ [5]	2.74	9.33	Torch
Li+ [6]	3.53	9.42	Torch

generating one stylized image. As we see, our method is the fastest and speeds up 22 times for the image size of 256×256 and 21 times for that of 512×512 when compared with the fastest state-of-the-arts [1]. We can thus conclude that our method is promising for real-time applications.

6.4. More detailed analysis

6.4.1 Behavior of balance weight γ during the training

We investigate the behavior of balance weight γ to verify that γ is adaptively updated to converge to an expected value.

Figure 11 illustrates how balance weight γ changes during the training on the Starry Night style. We see that γ is

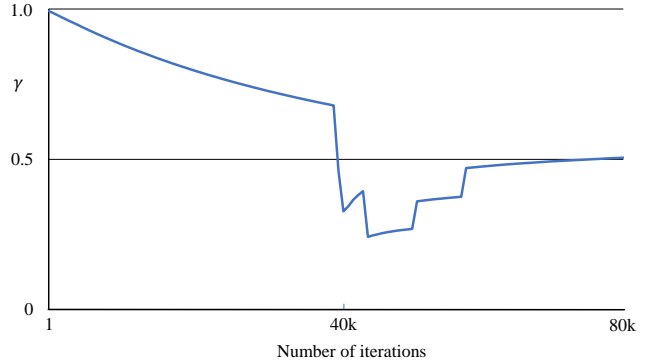


Figure 11: Behavior of γ during the training on the Starry Night style.

adaptively updated corresponding to the content and style losses. We remark that since we set $\alpha = 0.5$ in the loss function, γ is expected to be close to 0.5 after the training. At the beginning of training, the style loss \mathcal{L}_s is far larger than the content loss \mathcal{L}_c , resulting in γ far larger than 0.5 (close to 1.0). As the training proceeds, the network is gradually optimized, resulting γ close to 0.5 in the end of the training.

We observe that γ quickly decreases after one epoch (about 40k iterations). This can be explained as follows. After one epoch, the overfitting problem on the style image occurs since our network is trained using a single style image. Hence, the style loss quickly drops. As a result, the behavior of γ becomes different. Indeed, we observed that the style loss raised the overfitting problem through the validation phase. We thus stopped the training of the style subnet while kept updating the weights of the other subnets. As a result, the content loss decreased more quickly than the style loss. Then, γ was gradually recovered; its value became close to 0.5 in the end of training.

This evaluation confirms that γ gradually adapts to achieve the equal contributions of content and style in stylized images during the training thanks to our adaptive feature injection and concatenation. We remark that we observed similar behaviors of γ for other styles.

6.4.2 Effectiveness of feature injection

In this section, we evaluate the effectiveness of the introduction to the adaptive feature injection between the content subnet and the style subnet.

We compared our complete model with the model w/o feature injection (i.e., the model that disabled only the adaptive feature injection), which is shown in Fig. 12. Fig. 12 shows that the stylized images obtained by the complete model are in general more balanced in content and style than those by the model w/o feature injection. However, we

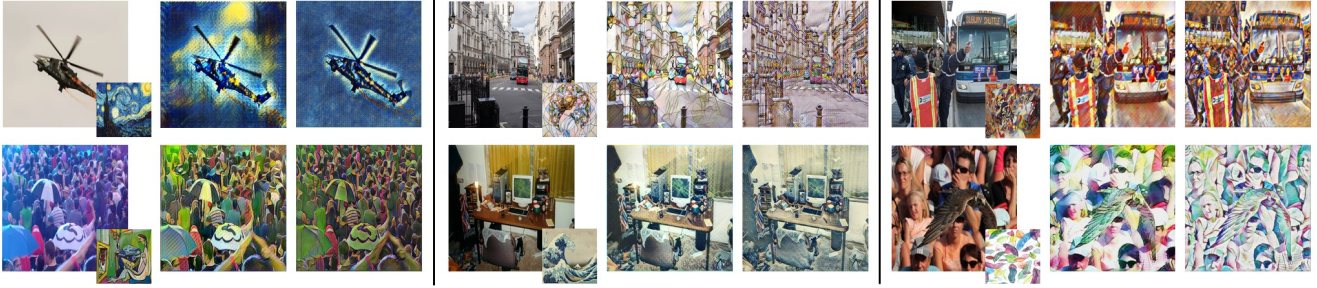


Figure 12: Visual comparison of the complete model and the model w/o feature injection. In each block, from left to right, a content image (large one) with a style (small one) is followed by outputs by the complete model and the model w/o feature injection. Note that all stylized images are with the size of 512×512 .

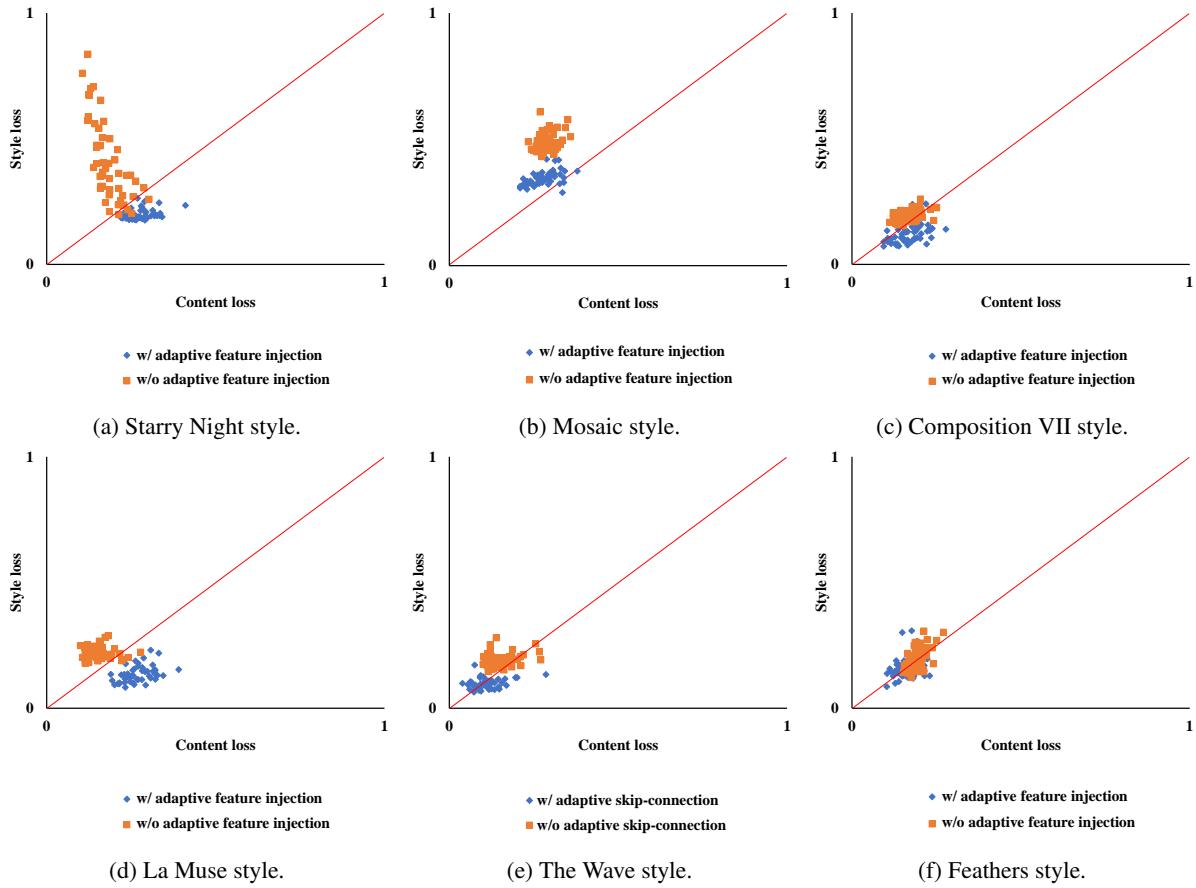


Figure 13: Loss distribution in each style obtained by the complete model and the model w/o feature injection. Red lines denote the balanced axis.

can see roughly global structure appearing in the synthesized images in Fig. 12 upper set (in particular, the leftmost which is with Starry Night style). This can be explained as follows. In general, the model w/o feature injection tends to preserve more content than style while the complete model does more style than content. This is because the feature

injection from the style subnet to the content subnet tries to reduce the style loss (see below). The feature injection at multiple layers employed in the content and style subnets helps to keep both global and local structure in rendering. As a result, global structure in stylized images such as the stroke in the Starry Night may sometimes become impres-

Table 7: Averages of *length* (smaller is better) and *balance* (larger is better) in the complete model (denoted by **complete**) and the model w/o feature injection (denoted by **w/o injection**).

Style	<i>length</i> (\Downarrow)		<i>balance</i> (\Uparrow)	
	complete	w/o injection	complete	w/o injection
Starry Night	0.34	0.46	2.12	1.35
Mosaic	0.45	0.57	1.80	1.03
Composition VII	0.21	0.26	3.61	3.21
La Muse	0.30	0.27	1.72	2.54
The Wave	0.15	0.24	5.39	3.15
Feathers	0.23	0.28	3.71	3.20
All together	0.28	0.35	3.06	2.41

sive.

We also compared the *length* and *balance* of stylized images (Table 7). We see that the complete model performs better both in *length* and *balance* than the model w/o feature injection. Table 7 also shows that employing adaptive feature injection improves both *length* and *balance* for each style (except for La Muse style). This indicates that adaptive feature injection is effective to improve not only the quality but also the balance in content and style of stylized images. With respect to the La Muse style, *length* of the complete model is comparable to that of model w/o feature injection, however *balance* is not the case. This can be explained as follows. The La Muse style follows Cubism and thus it is very unique. Because of this, the adaptive feature injection tends to keep more style to reflect the impression of this style.

Finally, we compare the loss distributions of 50 stylized images in each style (Fig. 13). We see that for all styles (except for the La Muse style) the loss distributions of the complete model appears more densely near the balanced axis and is closer to the origin than those of the model w/o feature injection for all styles. In the case of the Starry Night style (Fig. 13a), we see that the model w/o feature injection preserves much more content than the style because the loss distribution appears far above the balanced axis. This observation also holds true for the Mosaic style (Fig. 13b), the Composition VII (Fig. 13c), and the La Muse (Fig. 13d). By using adaptive feature injection, the complete model is able to reduce the style loss in stylized images (e.g., the Starry Night, the Mosaic, the Composition VII, the La Muse styles), compared to the model w/o feature injection. These observations indicate that the adaptive feature injection effectively improves to keep the balance in content and style of stylized images.

6.4.3 Effectiveness of combination weight α and balance weight γ

Here, we evaluate the necessity of combination weight α in Eq. (1) and balance weight γ in Eq. (4). In particular, we evaluate whether α plays the role of explicitly controlling the contribution ratio of the content and the style.

We generated stylized images using different values of α : $\alpha = 0.1, 0.3, 0.7, 0.9$. The results are illustrated in Fig. 14 where the complete model denotes the model using α and γ together while the model w/o γ denotes the model using α only (i.e., γ is disabled). Ideally, for smaller α , the style is more emphasized and results become more similar to those by Gatys+ [3]. For larger α , on the other hand, the content is more emphasized and results become more similar to those by Johnson+ [1]. We observe these in Fig. 14 and see that α of the complete model indeed controls the contribution ratio of the content and the style as we expected. However, we see that the model w/o γ is not the case. This observation suggests the necessity of both α and γ .

7. Conclusion

We presented an end-to-end two-stream network for balancing the content and style in stylized images. Our proposed method utilizes a deep FCN to preserve the semantic content and a shallow FCN to faithfully learn the style representation, whose outputs are adaptively feature injected and concatenated using the balance weight and fed into the decoder to generate stylized images. Our intensive experiments using six famous styles widely used in style transfer demonstrate the effectiveness of our proposed method against state-of-the-art methods in terms of balancing content and style. Furthermore, our proposed method outperforms the state-of-the-art methods in speed.

Our proposed method requires fine-tuning of parameters from an existing model to deal with different styles. This limits the applicability of our proposed method to multi-style transfer. Extending our proposed method so that it can deal with a large style dataset such as Wikiart or unseen styles is left for future work.

As an extension of image style transfer, the real-time video stylization methods are currently proposed [38, 39, 40]. Since our proposed method runs fast, we believe that it can be useful for real-time video stylization. Though video stylization is out of the scope of this paper, we applied our method in the frame-by-frame manner to several videos for video stylization demonstration. Fig. 15 shows some examples of stylized frames from a video. Our approach was able to stylize videos in real-time with the resolution 480×640 at 30 FPS or more. As we see, our method produces reasonable results for consecutive frames with varying appearance, meaning that the usage of our method for real-time video stylization is promising. We remark that we did not

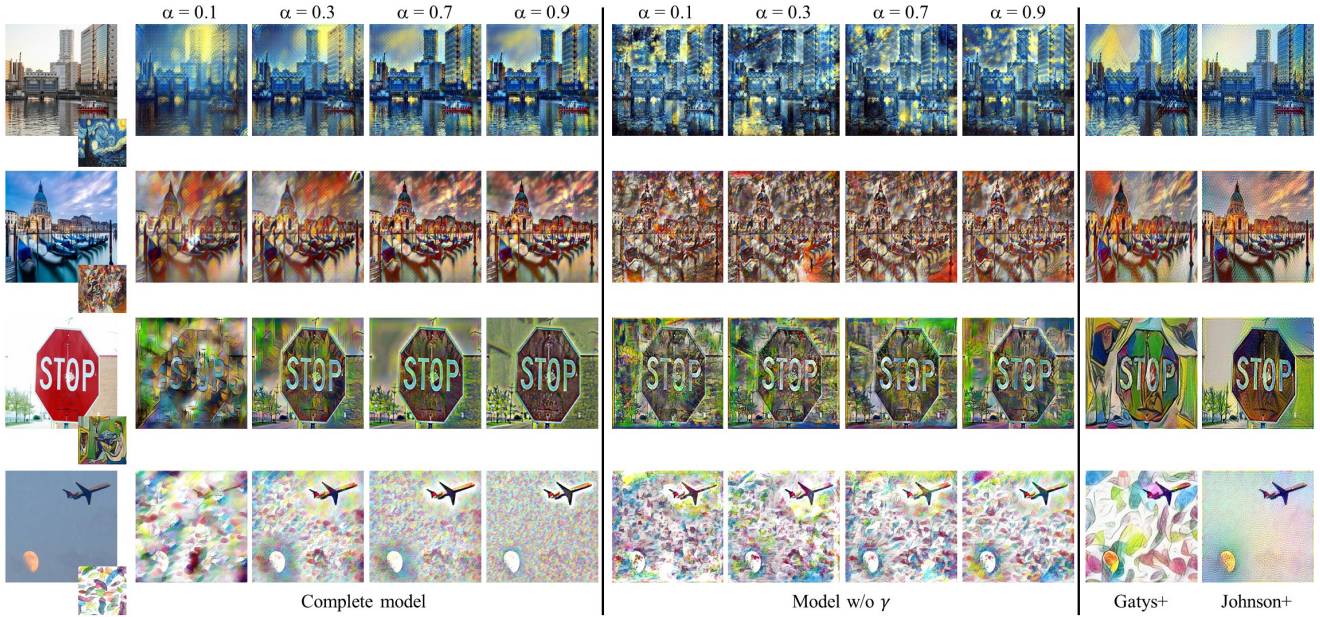


Figure 14: Example of stylized images by changing α from 0.1 to 0.9. Left-most column: the content image (large) and the style image (small). From left to right: the stylized image using various α . The last column shows results obtained by Gatys+ [3] and Johnson+ [1] for the reference.

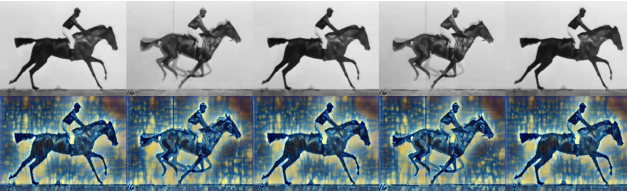


Figure 15: Examples of stylized video in real-time using the "Starry night" style. We use the video of Eadweard Muybridge "The horse in motion" (1878) as the content input. Our model processes every frame independently without any post-processing. Video resolution is 480×640 at 30 FPS.

use either temporal regularization or post-processing. Different from image style transfer, real-time video stylization needs to pay attentions to the temporal consistency among adjacent video frames. Incorporating the temporal consistency into our method for real-time video stylization is left for our future work.

Acknowledgements

This is a pre-print of an article published in Machine Vision and Applications. The final authenticated version is available online at: <https://doi.org/10.1007/s00138-020-01086-1>

This work was in part supported by JST CREST (Grant No. JPMJCR14D1). The authors are thankful to Dr. Trung-

Nghia Le for his valuable comments on this work.

References

- [1] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *ECCV*, 2016. 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13, 15, 16
- [2] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *ICCV*, 2017. 1, 2, 3, 4, 8, 9, 10, 11, 13
- [3] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *CVPR*, 2016. 1, 2, 3, 4, 5, 8, 9, 10, 11, 13, 15, 16
- [4] L. Sheng, Z. Lin, J. Shao, and X. Wang, "Avatar-net: Multi-scale zero-shot style transfer by feature decoration," in *CVPR*, 2018. 1, 2, 3, 4, 8, 9, 10, 13
- [5] T. Q. Chen and M. Schmidt, "Fast patch-based style transfer of arbitrary style," in *NIPS*, 2016. 1, 2, 3, 4, 5, 8, 9, 10, 13
- [6] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Universal style transfer via feature transforms," in *NIPS*, 2017. 1, 2, 3, 8, 9, 10, 13
- [7] J. E. Kyprianidis, J. Collomosse, T. Wang, and T. Isen-berg, "State of the "art": A taxonomy of artistic

- stylization techniques for images and video,” *IEEE Transactions on Visualization and Computer Graphics*, 2013. 1
- [8] M. Ashikhmin, “Synthesizing natural textures,” in *Symposium on Interactive 3D Graphics*, 2001. 1, 2
- [9] A. A. Efros and W. T. Freeman, “Image quilting for texture synthesis and transfer,” in *SIGGRAPH*, 2001. 1, 2
- [10] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, “Neural style transfer: A review,” *IEEE Transactions on Visualization and Computer Graphics*, 2019. 1, 4, 11
- [11] F. Luan, S. Paris, E. Shechtman, and K. Bala, “Deep photo style transfer,” in *CVPR*, 2017. 1, 3
- [12] R. Mechrez, E. Shechtman, and L. Zelnik-Manor, “Photorealistic style transfer with screened poisson equation,” in *BMVC*, 2017. 1, 2, 3
- [13] S. Azadi, M. Fisher, V. Kim, Z. Wang, E. Shechtman, and T. Darrell, “Multi-content gan for few-shot font style transfer,” in *CVPR*, 2018. 1, 2, 4
- [14] D. Kotovenko, A. Sanakoyeu, P. Ma, S. Lang, and B. Ommer, “A content transformation block for image style transfer,” in *CVPR*, 2019. 1, 2, 4
- [15] D. Y. Park and K. H. Lee, “Arbitrary style transfer with style-attentional networks,” in *CVPR*, 2019. 1, 2
- [16] A. Sanakoyeu, D. Kotovenko, S. Lang, and B. Ommer, “A style-aware content loss for real-time hd style transfer,” in *ECCV*, 2018. 1, 2, 4
- [17] X. Wang, G. Oxholm, D. Zhang, and Y.-F. Wang, “Multimodal transfer: A hierarchical deep convolutional neural network for fast artistic style transfer,” in *CVPR*, 2017. 1, 2, 3, 4, 5, 8
- [18] Y. Zhang, Y. Zhang, and W. Cai, “Separating style and content for generalized style transfer,” in *CVPR*, 2018. 2
- [19] Y. Li, M.-Y. Liu, X. Li, M.-H. Yang, and J. Kautz, “A closed-form solution to photorealistic image stylization,” in *ECCV*, 2018. 2
- [20] D. M. Vo, T. N. Le, and A. Sugimoto, “Balancing content and style with two-stream fcns for style transfer,” in *WACV*, 2018. 2
- [21] D. J. Heeger and J. R. Bergen, “Pyramid-based texture analysis/synthesis,” in *SIGGRAPH*, 1995. 2
- [22] S. Li, X. Xu, L. Nie, and T.-S. Chua, “Laplacian-steered neural style transfer,” in *ACM-MM*, 2017. 3
- [23] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky, “Texture networks: Feed-forward synthesis of textures and stylized images,” in *ICML*, 2016. 3
- [24] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” in *ICLR*, 2016. 3
- [25] V. Dumoulin, J. Shlens, and M. Kudlur, “A learned representation for artistic style,” in *ICLR*, 2017. 3, 6, 8
- [26] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, “Instance normalization: The missing ingredient for fast stylization,” in *ICML*, 2016. 3, 6, 7
- [27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *NIPS*, 2014. 4
- [28] C. Li and M. Wand, “Precomputed real-time texture synthesis with markovian generative adversarial networks,” in *ECCV*, 2016. 4
- [29] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *ICLR*, 2015. 4, 5, 8
- [30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, 2015. 4, 8
- [31] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *ICML*, 2010. 4, 6, 7
- [32] A. Mahendran and A. Vedaldi, “Understanding deep image representations by inverting them,” in *CVPR*, 2015. 4
- [33] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *ICLR*, 2015. 5, 8
- [34] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016. 6
- [35] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *ICML*, 2015. 6
- [36] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd ed. New York, NY, USA: Cambridge University Press, 2012. 7
- [37] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *ECCV*, 2014. 8

- [38] C. Gao, D. Gu, F. Zhang, and Y. Yu, “Reconet: Real-time coherent video style transfer network,” in *ACCV*, 2018. 15
- [39] H. Huang, H. Wang, W. Luo, L. Ma, W. Jiang, X. Zhu, Z. Li, and W. Liu, “Real-time neural style transfer for videos,” in *CVPR*, 2017. 15
- [40] W. Li, L. Wen, X. Bian, and S. Lyu, “Evolver: Evolutionary constrained adversarial learning for video style transfer,” in *ACCV*, 2018. 15