Pacific Graphics 2018
H. Fu, A. Ghosh, and J. Kopf
(Guest Editors)

Volume 37 (2018), Number 7

Controlling Stroke Size in Fast Style Transfer with Recurrent Convolutional Neural Network

Lingchen Yang[1] Lumin Yang[1] Mingbo Zhao[2] Youyi Zheng[1]†

[1] State Key Lab of CAD & CG, Zhejiang University
[2] College of Information Science and Technology, Donghua University

Abstract

Controlling stroke size in Fast Style Transfer remains a difficult task. So far, only a few attempts have been made towards it, and they still exhibit several deficiencies regarding efficiency, flexibility, and diversity. In this paper, we aim to tackle these problems and propose a recurrent convolutional neural subnetwork, which we call recurrent stroke-pyramid*, to control the stroke size in Fast Style Transfer. Compared to the state-of-the-art methods, our method not only achieves competitive results with much fewer parameters but provides more flexibility and efficiency for generalizing to unseen larger stroke size and being able to produce a wide range of stroke sizes with only one residual unit. We further embed the* recurrent stroke-pyramid *into the Multi-Styles and the Arbitrary-Style models, achieving both style and stroke-size control in an entirely feed-forward manner with two novel run-time control strategies.*

CCS Concepts

•***Computing methodologies*** → ***Neural networks; Image processing;***

1. Introduction

Rendering an image in the style of another one is a long-standing problem [GG01, SS02, KEBK05]. Inspired by the deep learning algorithms, the seminal work of Gatys et al. [GEB15b, GEB15a] shows that a pre-trained convolutional neural network (CNN) can be used as a feature extractor to achieve this goal. The key idea is to optimize an image with a gradient descend method, so as to match the feature statistics of it to that of the style for texture synthesis, or both the style and the content for image stylization. However, this algorithm is extremely slow in terms of real-time application, as it needs hundreds of iterations to converge every time.

To resolve the problem, significant efforts have been devoted to speeding up neural style transfer. Ulyanov et al. [ULVL16] and Johnson et al. [JAFF16] attempt to train a feed-forward neural network that stylizes the input image in a single forward pass, which turns out to be three orders of magnitude faster than the method of [GEB15b, GEB15a]. Nonetheless, the flexibility is lost: a style transfer network is tied to one specific style. Surprisingly, Ulyanov et al. [DSK16] discover that specializing affine transformation parameters to each specific style is sufficient to model a style. They successfully incorporate different styles into only one generator with negligible extra parameters introduced. There are also many other methods [LFY*17a, ZD17, CYL*17] proposed to address this problem, but they are all restricted to a fixed number of styles. Later on, the works of [HB17, GLK*17, LFY*17b, SLSW18] introduce algorithms to transfer arbitrary style through only one single model. In short, these Fast Style Transfer algorithms can be classified into three categories: One-Style model [ULVL16, JAFF16, UVL17, LW16], Multi-Styles model [DSK16, LFY*17a, ZD17, CYL*17] and Arbitrary-Style model [HB17, GLK*17, LFY*17b, SLSW18].

The above methods don't take stroke-size control into account, which is an essential perceptual factor in paintings. Gatys et al. [GEB*17] first design a method to produce large stroke size for the high-resolution image, which can be used to change the stroke size but is apparently very slow and inflexible. Recently, Jing et al. [JLY*18] propose a module named *stroke-pyramid* to continuously control the stroke size of the generated images in real time. However, this method suffers from the following drawbacks: (1) the number of stroke sizes is fixed, and their method cannot adapt to new size without training the newly-introduced parameters; (2) the method of [JLY*18] needs a lot of parameters when a wide range of stroke sizes are needed, which is very wasteful. What's more, it remains unclear how effective this kind of structure is when used in Multi-Styles model or even Arbitrary-Style model.

In this work, we develop a novel style transfer method to tackle the aforementioned problems. Our work stems from the intuition that in *stroke-pyramid* [JLY*18], different residual units might share much degree of computation. However, this sharing will be thrown away by training different residual units as in [JLY*18].

† Corresponding author

Figure 1. An overview of our work. **(a)** The proposed *recurrent stroke-pyramid* described in Section 3.1. **(b)** The *recurrent stroke-pyramid* can generalize well to unseen larger stroke size. The picture above represents the largest stroke size during training while below reflects unseen larger stroke size during testing. **(c)** It can produce a wide range of stroke sizes with only one residual unit. **(d)** Spatially controlling both style and stroke size in a totally feed-forward manner. The picture above only uses style spatial control while below uses both two. **(e)** and **(f)** When embedded into Arbitrary-Style model, it can control the stroke size of unseen styles without any image processing before or after one-shot forwarding.

Thus, we design a recurrent convolutional neural subnetwork to replace the original *stroke-pyramid*, which we call *recurrent stroke-pyramid*. Consequently, this kind of recurrent computation can automatically increase the receptive field of the generator without introducing extra parameters, making it flexible to generate larger unseen stroke sizes than the training set. Besides, it can save a lot of parameters when a wide range of stroke sizes are needed. Finally, we extend our proposed method to the Multi-Styles and the Arbitrary-Style models to incorporate different styles into one single model while the stroke size of each style can be controlled without any image processing before or after one-shot forwarding.

Our contributions are threefold: (1) We re-analyze the function of each residual unit used in [JLY*18] and propose a novel recurrent convolutional neural subnet to control stroke size in Fast Style Transfer. (2) Experiments demonstrate the high efficiency and flexibility of our method for parameters reduction, ease of training, generalization to unseen larger stroke size, and the capability to produce a wide range of stroke sizes only with one residual unit. (3) We newly design the network architecture to embed this pyramid structure into the Multi-Styles and the Arbitrary-Style models, and present two novel run-time control strategies to concurrently control the style and the stroke size in an entirely feed-forward manner.

2. Background and related work

Below we briefly introduce the background and relevant methodologies in Fast Style Transfer.

Style Normalization. Style normalization (SN) is a domain transformation technique applied to each sample image independently, which is commonly used in Fast Style Transfer [UVL17, DSK16, HB17, GLK*17, LFY*17b, SLSW18]. It transforms the current domain $\mathcal{D}_c$ back to the common domain $\tilde{\mathcal{D}}$ and then to the style domain $\mathcal{D}_s$ in the feature space of a CNN, where $\tilde{\mathcal{D}}$ contains the basic information, e.g., orientation, but dispels several details, e.g., amplitude. Specifically, let $x \in \mathbb{R}^{C\times(H\times W)}$ denote the input to the SN layer, then the output of it can be described as:

$$y = W_s W_c^{-1}(x-\mu_c)+\mu_s, \tag{1}$$

where W_c, $W_s \in \mathbb{R}^{C\times C}$ are transformation matrices respectively for $\mathcal{D}_c$ and $\mathcal{D}_s$; μ_c, $\mu_S \in \mathbb{R}^C$ represent the channel-wise mean. It should be noted that W_c and μ_c are derived from x while W_s and μ_s from data [UVL16, DSK16] or a style encoder [HB17, GLK*17, LFY*17b, SLSW18].

Among these methods, Instance Normalization (IN) [UVL16] is first proposed to improve the stylization quality, but it only normalizes different images into one single style. When multiple styles are considered, W_s and μ_s should be augmented to independent pairs with each one conditional on a specific style, which is known as Conditional Instance Normalization (CIN) proposed by Dumoulin [DSK16]. In addition, given W_s and μ_s computed from a style encoder, the whole network will be able to transfer arbitrary style according to the works in [HB17, GLK*17]. Later on, Li et al. [LFY*17b] discover that a well-designed SN layer, which they call Whitening and Coloring Transformation (WCT), can be effectively used in the auto-encoder architecture to achieve zero-shot style transfer. Very recently, Sheng et al. [SLSW18] introduce the Style Decoration (SD) module to improve WCT for better quality.

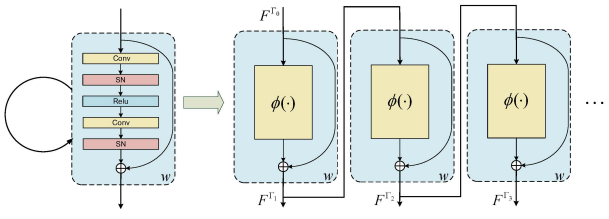


Figure 2. The *recurrent stroke-pyramid* receives the input feature map F^{Γ_0} and output a sequence of feature maps $\{F^{\Gamma_i}\}_{i=1}^{R}$ after R steps, during which the parameters w are shared.

However, these auto-encoder-based methods will find it hard to recover fine-grained style patterns due to the downsample operations.

Stroke-size Control. Stroke size plays an essential part in paintings, which measures how big a corresponding brush is used and how much distance it moves. Thus, stroke size can be defined as the average scale of the similar-sized stroke texons [JLY*18,Jul81,ZGWX05] in an area. There are three primary requirements for stroke-size control: (1) generating semantically distinguishable images no matter which stroke size is used; (2) continuously controlling the stroke size in a feed-forward manner; (3) preserving the stroke consistency [JLY*18] that the generated images of different stoke sizes should exhibit consistent stroke orientation, configuration, etc.

In the Fast Style Transfer literature, the One-Style model [ULVL16,JAFF16] can train multiple models to obtain different stroke sizes. However, it is very uneconomic and inflexible. Wang et al. [WOZW17] design a hierarchical convolutional neural network to paint large strokes in high-resolution images. However, it inherently entails multiple models for different stroke sizes. Furthermore, the works in [ULVL16,JAFF16,WOZW17] dissatisfy requirement (2) since one model only produce one stroke size. Though Some SN methods [HB17, GLK*17, LFY*17b, SLSW18] can achieve arbitrary-style transfer with one single model, they are not good at controlling stroke size when the receptive field is fixed (see Section 4.2.2). Moreover, they need image processing to achieve different stroke sizes, which is very inflexible. To resolve this dilemma, Jing et al. [JLY*18] recently discover that the generator with larger receptive field can produce larger stroke size and propose a module named *stroke-pyramid* to endow the network with adaptive receptive fields. It consists of several stroke branches, namely separate residual units, with each one having a larger receptive field than the previous one. As a result, this method perfectly meets the requirements with only one model, becoming the state-of-the-art method.

Nevertheless, another problem occurs that compared with a network without *stroke-pyramid*, the work in [JLY*18] requires $L\times 2\times(9C^2+2C)$ additional parameters, where C is the number of residual units' channels and L the number of stroke branches. Furthermore, if C is large (e.g., $C=128$), overall parameters will come from the residual units and increase linearly with L. Therefore, it seems wasteful to apply their method to wide-ranging stroke sizes. Besides, their approach can not adapt to new stroke size without training the newly-introduced parameters. On the other hand, since they only focus on the One-Style model, it's unclear how to extend their method to the Multi-Styles and the Arbitrary-Style models. Note that our work is intrinsically different to [JLY*18] as we propose a novel technique to control the stroke size, and newly design the network architecture and loss function to explore the effectiveness of it for diverse styles in one single model.

3. The Method

3.1. Recurrent Stroke-Pyramid

Our work stems from the intuition that in *stroke-pyramid*, different residual units have the similar function: fine-tune the feature map from the previous one to let the stroke decoder produce larger stroke size, which means they might share much degree of computation. Therefore, training separate residual units for different stroke sizes will leave this sharing vulnerable to be broken.

Taking this into consideration, we design the *recurrent stroke-pyramid* which is a recurrent convolutional neural network and inserted in the encoder-decoder architecture (see Section 3.2 for details). The main part of it is a residual unit [HZRS16a] similar to the proposed method of [HZRS16b], where a kind of SN layer is substituted for the Batch Normalization [IS15] layer. This structure has two advantages. First, the parameters are shared across the whole procedure. Second, inserting convolutional computation into the recurrent neural network can endow the generator with adaptive receptive fields, which is the key to controlling the stroke size according to [JLY*18]. In each step, the input to the residual unit comes from the output feature map in the previous step (or the input feature map if this is the first step). Thus, after R steps, our *recurrent stroke-pyramid* will output a sequence of R feature maps. An illustration of the *recurrent stroke-pyramid* can be as seen in Figure 2. Here, let F^{Γ_0} denote the input feature map to the module and define convolution, style normalization and Relu nonlinearity as $\phi_C(\cdot)$, $\phi_{SN}(\cdot)$ and $\phi_f(\cdot)$ respectively, then the output feature map $F^{\Gamma_i}(i=1,\cdots,R)$ in the ith step that corresponds to the ith bigger stroke size Γ_i, can be computed by:

$$\begin{aligned} F^{\Gamma_i} &= F^{\Gamma_0}+\sum_{j=0}^{i-1}\phi_{SN}(\phi_C(\phi_f(\phi_{SN}(\phi_C(F^{\Gamma_j}))))) \\ &= F^{\Gamma_0}+\sum_{j=0}^{i-1}\phi(F^{\Gamma_j};w) \end{aligned} \tag{2}$$

where w denotes the parameters of the residual unit. In this formula, the residual unit will fine-tune the feature map $F^{\Gamma_{i-1}}$ by adding the term $\phi(F^{\Gamma_{i-1}};w)$ in the ith step. Having observed that, we expect this addition will consistently tell the decoder to produce larger stroke size than $F^{\Gamma_{i-1}}$ does. Besides, this recurrent computation can enforce stroke consistency (see Section 2) and automatically increase the receptive field of the whole network without introducing extra parameters, making it possible to generate larger unseen stroke size during testing flexibly. It can also save a lot of parameters when a wide range of stroke sizes is needed, which will wastefully induce many separate residual units to be trained in [JLY*18].

3.2. Architecture

Figure 3 shows our proposed architecture. It consists of four parts: encoder, *recurrent stroke-pyramid*, fusion module and decoder. The convolution layers are all followed by a kind of SN layer to control the style. The padding mode is set to reflect-padding. All the activation functions are Relu nonlinearity except the sigmoid function in the last layer. The encoder is stacked by three convolution layers and two residual units while the decoder consists of two up-sampling layers [ULVL16]. Notice that the style control and the stroke-size control are mutually independent thanks to the separation of the convolution and the normalization operations.

SN Layer. For single-style transfer, we use the IN layer. For multi-styles transfer, the CIN [DSK16] layer is applied to our network. For arbitrary-style transfer, we use the similar style normalization method as in [GLK*17], where the parameters W_s and μ_s for each style are computed from the Inception v3 [SVI*16]. We formalize W_s and W_c as diagonal matrices whose diagonal elements

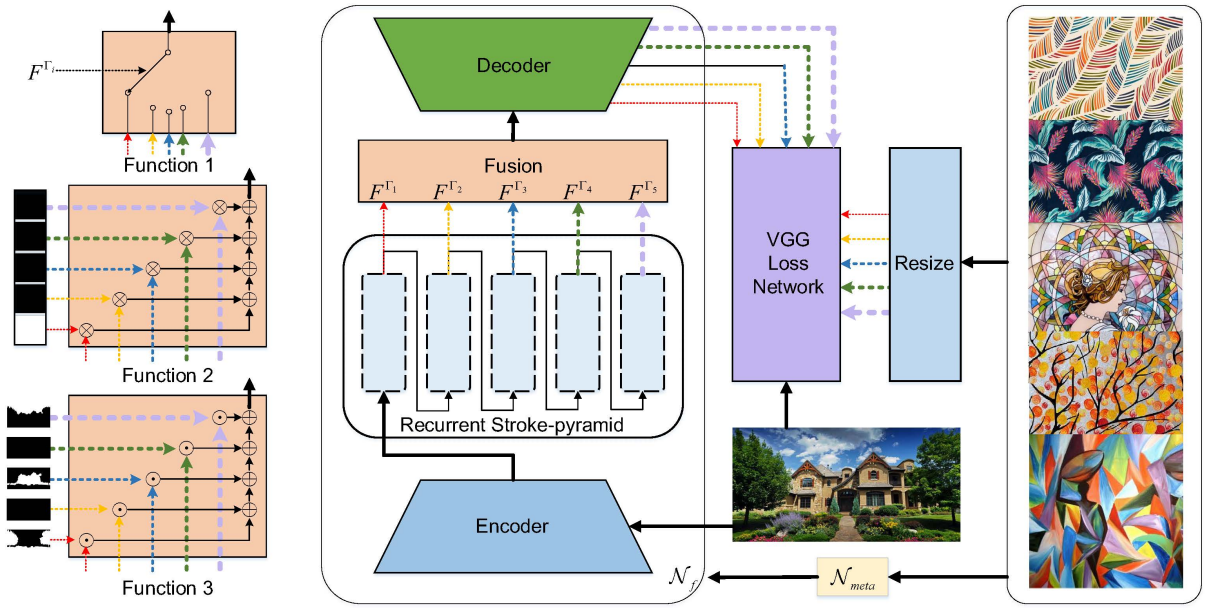


Figure 3. An overview of our algorithm. In this picture, the recurrent time R is five. The *recurrent stroke-pyramid* receives the input feature map from the encoder and outputs a sequence of feature maps. Then the fusion model will receive it and use one of the three functions to generate the output feature map for the decoder. During training, each style image will be resized to R different sizes and each size corresponds to one specific stroke size. $\mathcal{N}_{meta}$ will update the parameters of $\mathcal{N}_f$ based on current style. Details are described in Section 3.2.

correspond to channel-wise standard variance [DSK16, GLK*17], mainly for high efficiency and also good stylization quality.

Recurrent Stroke-pyramid. According to Section 3.1, this module will receive the feature map from encoder and output a sequence of feature maps after several recurrent steps. Each feature map has a larger receptive field than that produced in previous recurrent step through 2 newly increased convolution operations. To let the later-produced feature map contain larger stroke size information, the corresponding style image should be used for training, which is achieved by the resizing method. During testing, its recurrent time can be extended to be longer than during training, making it possible to produce larger unseen stroke size.

Fusion Module. This module has three functions, as illustrated in the pink regions of Figure 3. Function 1: during training, it acts as a multiplexer that chooses one feature map from input sequence for decoder based on current style stroke size Γ^i in each iteration. Function 2: if given a list of convex weights $\{\hat{\alpha}^i\}_{i=1}^R$ that $\sum_{i=1}^R \hat{\alpha}^i = 1$, it will output fused feature map $F^{\tilde{\Gamma}} = \sum_{j=1}^R \hat{\alpha}^j F^{\Gamma_j}$. Function 3: if given a list of masks $\{M^i\}_{i=1}^R$ indicating the spatial correspondence between content regions and stroke sizes, it will spatially control the stroke size within one image $F^{\tilde{\Gamma}} = \sum_{j=1}^R M^j \odot F^{\Gamma_j}$, where $\odot$ is a simple mask-out operation.

Run-time Control Strategies. The Function 2 and 3 of the fusion module can achieve continuous control and spatial control of the stroke size respectively [JLY*18]. When these functions are executed synchronously with the style control strategies [DSK16, HB17], there emerge two novel run-time strategies. Strategy 1: concurrently interpolating between styles and stroke sizes. Strategy 2: concurrently executing style and stroke-size spatial control. The results and details can be found in 4.3.2.

3.3. Loss Function

Content Loss. Based on the works in [GEB15b, ULVL16, JAFF16], the content loss is formulated as the Euclidean distance between the content target I_c and the generated image I in the high-level feature space of the VGG network [SZ14]. Let $\Psi_l(x)$ denote the feature map of shape $C_l \times H_l \times W_l$ in the lth layer of the loss network Ψ with input image x. The content loss $\mathcal{L}_c$ is defined as:

$$\mathcal{L}_c(I, I_c) = \frac{1}{2C_l H_l W_l} \|\Psi_l(I) - \Psi_l(I_c)\|_2^2. \quad (3)$$

Style loss. Let $G(\Psi_l(x))$ denote the Gram matrix of $\Psi_l(x)$ that represents the style statistics [GEB15a], then the style loss $\mathcal{L}_s$ is the Frobenius norm of the difference between $G(\Psi_l(I))$ and $G(\Psi_l(I_s))$, where I_s indicates the style target. During training, for one feature map F^{Γ_i} selected by the fusion module, the style image $I_s^{\Gamma_i}$ of stroke size Γ_i is used. Then, the style loss for the output image I^{Γ_i}, generated by feeding F^{Γ_i} to the decoder, can be described as:

$$\mathcal{L}_s(I^{\Gamma_i}, I_s^{\Gamma_i}) = \sum_{l \in \{l_s\}} \alpha_l \frac{1}{4C_l^2 N_l^2} \left\| G(\Psi_l(I^{\Gamma_i})) - G(\Psi_l(I_s^{\Gamma_i})) \right\|_2^2, \quad (4)$$

where $I_s^{\Gamma_i}$ is obtained by resizing I_s based on Γ_i, $N_l = H_l \times W_l$, $\{l_s\}$ is the set of loss network layers used to compute the style loss, and α_l is the weight for the lth layer. During training, the weight for high-level layers will increase linearly with current stroke size to enhance the large pattern.

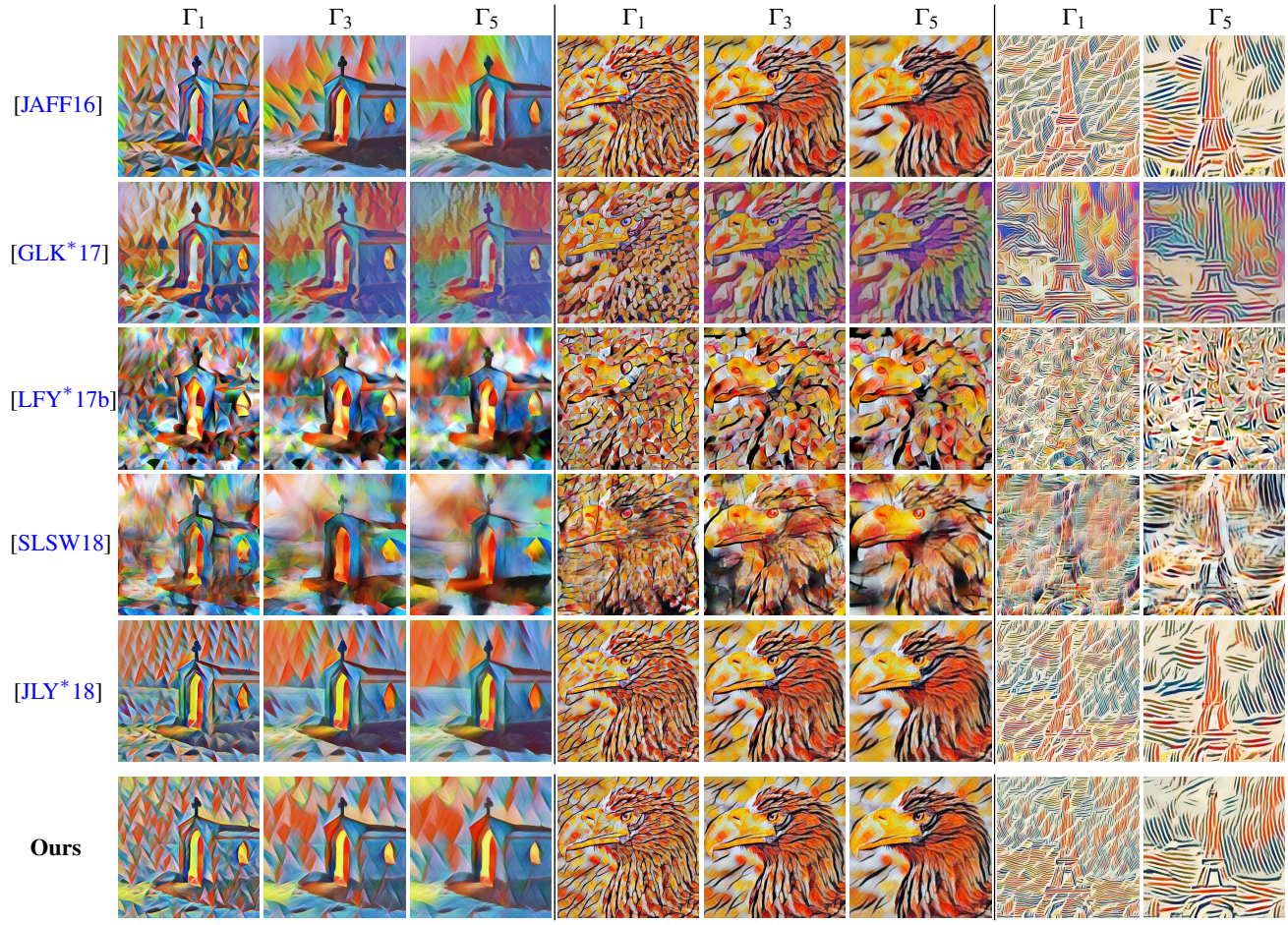


Figure 4. Visual comparisons between our *recurrent stroke-pyramid* and [JAFF16, GLK*17, LFY*17b, SLSW18, JLY*18]. From right to left, the style images are Figure 14 (g) (i) (a), content images are Figure 13 (g) (e) (i) in the Appendix. Note that [JAFF16] entails multiple models while ours and [JLY*18, GLK*17, LFY*17b, SLSW18] can achieve different stroke sizes with only one single model.

Total Loss. In order to transfer multiple or even arbitrary style, different styles should have their own W_s and μ_s, as discussed in Section 2. We use $\mathcal{N}_{meta}$ to abstractly represent the mapping from style image I_s to its corresponding parameters w_θ. The total loss for I^{Γ_i} can be then written as:

$$\mathcal{L}_{\Gamma_i} = \sum_{I_s \in \{I_s\}} \lambda_c \mathcal{L}_c(I^{\Gamma_i}, I_c) + \lambda_s \mathcal{L}_s(I^{\Gamma_i}, I_s^{\Gamma_i}) + \lambda_{tv} \mathcal{L}_{tv}(I^{\Gamma_i}), \quad (5)$$

where $I^{\Gamma_i} = \mathcal{N}_f(I_c; w, \Gamma_i)$; $\mathcal{N}_f$ denotes the generator; $w = (w_d, w_\theta)$, w_d represents the parameters shared by different styles, $w_\theta = \mathcal{N}_{meta}(I_s)$; total variation loss $\mathcal{L}_{tv}$ is added for piece-wise smoothness; λ_c is set to 1.0 while λ_s and λ_{tv} are left as hyper-parameters. The Algorithm 1 in the Appendix details our training strategy.

4. Experiments

4.1. Details

We train our proposed network using MS-COCO [LMB*14] as content images with a batch size of 8. They are resized and randomly cropped to 256×256 patches. We let the recurrent time be 5 as default and use 5 scales for each style image to achieve different stroke sizes. During training, the pre-trained VGG-16 network [SZ14] is chosen as the loss network and relu4_2 is used as the content layer while {relu1_1, relu2_1, relu3_1, relu4_1, relu5_1} as the style layers where {relu3_1, relu4_1, relu5_1} are set to be high-level. Adam [KA15] optimizer is applied with a learning rate of 1.0×10^{-3}. All of our experiments are implemented on Tensorflow[†]. The content and the style images used in the experiments can be found in Figure 13 and Figure 14 in the Appendix.

4.2. Evaluation of Our Recurrent StrokePyramid

In this section, we focus on the evaluation of the *recurrent stroke-pyramid*. We use three kinds of evaluation metric: quantitative comparison, qualitative comparison and flexibility comparison. IN is used in our network for a fair comparison with [JLY*18].

[†] https://www.tensorflow.org

Table 1. Differences between our approach and other methods.

	Parameters(5 sizes)	Requirments	Receptive Field
[JAFF16]	8.39M	×	fixed
[GLK*17]	3.82M	×	fixed
[LFY*17b]	34.24M	×	fixed
[SLSW18]	7.01M	×	fixed
[JLY*18]	2.27M	✓	adjustable
Ours	**1.09M**	✓	**adjustable**

4.2.1. Quantitative Comparison

In order to see what is being traded off by incorporating different residual units into only one in a recurrent manner, we use the same training strategy to train our proposed model and the model in [JLY*18], then compare them regarding the convergence speed and the final average loss of different stroke sizes. We set the number of stroke branches in [JLY*18] to be 5 for consistency with our recurrent time. Also, we let their encoders and decoders respectively be the same structures. Table 1 shows the parameters usage comparison between these two models. In this experiment, the parameters of the model in [JLY*18] are more than twice that of ours.

Top two figures in the Figure 5 compare the learning dynamics for each stroke size between two models. The losses are all averaged over 32 random batches of content images and 10 styles. By visual inspection, we can observe that for each stroke size, ours converges as quickly as [JLY*18] with respect to both the style and the content losses. In order to quantify this observation, we show the final loss difference for each stroke size between these two models below. Roughly, the difference is negligible since our proposed model's average final content loss over 5 different stroke sizes is 0.57% lower than [JLY*18] while average final style loss difference is even less (0.51% higher). What's more, as stroke size becomes larger, the style loss difference stabilizes (see stroke size Γ_4 and Γ_5), which is about 2.7% higher, also very insignificant. Therefore, ours can achieve quantitatively competitive results while improving the parameter-usage efficiency.

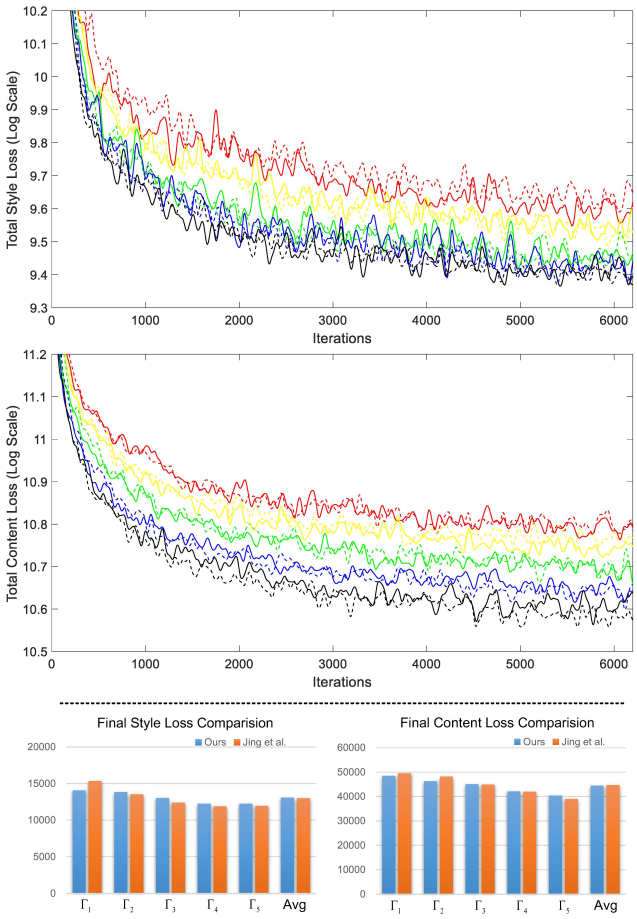


Figure 5. Top two figures compares the training procedure. For one specific stroke size, it is represented by two curves with the same color (red - Γ_1, yellow - Γ_1, green - Γ_3, purple - Γ_4, black - Γ_5). The full curve represents our proposed method while the dashed curve the method of [JLY*18]. Below is the final average loss comparison for each stroke size.

4.2.2. Qualitative Comparison

Figure 4 and Figure 8 show several samples of ours and [JLY*18], it is evident that the two models produce closely similar results. Specifically, they can generate semantically clear images with wide-ranging stroke sizes and preserve the stroke consistency that all the strokes change together only in size, namely satisfying the requirement (1) and (3) (Section 2). Note that we use much fewer parameters than [JLY*18] to achieve comparable visual results. Furthermore, our approach is more flexible, as shown in 4.2.3.

We also visually compare our method with [JAFF16, GLK*17, LFY*17b, SLSW18] in Figure 4. To obtain multiple stroke sizes, [JAFF16] needs to wastefully train multiple networks while Arbitrary-Style models [GLK*17, LFY*17b, SLSW18] should first resize input style images to different scales. As we can see, [JAFF16] will collapse the stroke consistency due to the separate learning of different models (e.g., the inconsistent stroke orientations on the left of the house in columns 1 through 3). The style encoder in [GLK*17] can't correctly parse the style patterns and result in unwanted ones when the stroke size mismatches its fixed and limited receptive field (e.g., the conspicuous discrepancy in overall hue between the original style and the stylized images with Γ_3 and Γ_5). Since [LFY*17b] and [SLSW18] employ an auto-encoder structure, when the stroke size is tiny, the decoder fails to completely reconstruct the subtle details due to the pooling layers and generate semantically vague images (see the stylized images with Γ_1, especially in column 7). Furthermore, WCT [LFY*17b] tends to generate unseen patterns (e.g., the distorted patterns in columns 1 through 3, 7 through 8) while SD [SLSW18] will collapse the stroke consistency (e.g., the style layout of the background varies with different stroke sizes), though they can holistically match the second-order statistics of arbitrary style and achieve high rendering quality generously. SD sometimes destroys the local continuousness of several style patterns (e.g., the long line patterns in column 8 are chaotic) due to the reassembling process in SD [SLSW18]. Moreover, these methods can't achieve continuous or spatial con-

trol of stroke size with economical processing cost, and they need much more parameters than ours. Table 1 summaries the above comparisons.

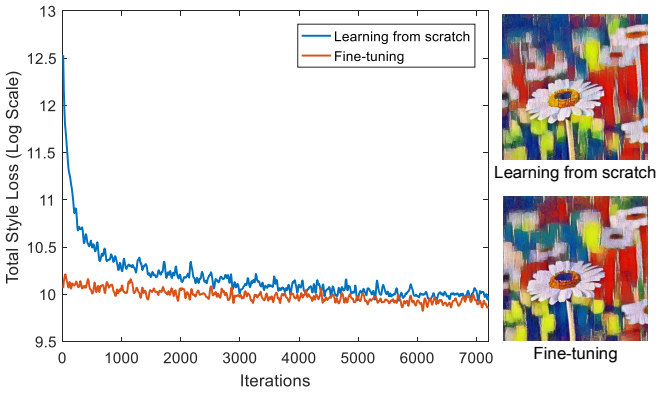


Figure 6. Comparisons between fine-tuning and learning from scratch in terms of style loss curve and quality.

Figure 7. Our *recurrent stroke-pyramid* generalizes well to unseen stroke size without any fine-tuning. Emphasis (red box) has been added on the results for comparison purpose.

4.2.3. Flexibility Comparison

Learning to Produce Larger Unseen Stroke Size. In order to produce unseen larger stroke size, [JLY*18] should retrain the network by incrementally learning a new branch on the top of *stroke-pyramid*. Although this method achieves faster convergence than training the whole network from scratch, it still needs hundreds of iterations to converge [JLY*18]. It should be noted that ours can achieve this goal by lengthening the recurrent time during testing. This is because the decoder learns to be sensitive to the changes in its input feature map caused by different recurrent times. The longer the recurrent time is, the larger stroke size should be achieved. To quantify this observation, we first augment a new larger stroke size upon the pre-trained network by fine-tuning it using the proposed training strategy (see the Algorithm 1 in the Appendix). Then, we compare the loss curve and stylization quality with that of optimizing the network from scratch (Figure 6). While achieving comparable results, fine-tuning nearly converges since the beginning, which indicates the network generalizes well to the larger unseen stroke size. We also let it produce unseen stroke size without any fine-tuning (Figure 7).

Producing a Wide Range of Stroke Sizes. In this case, our *recurrent stroke-pyramid* only needs to lengthen the recurrent time to be much longer (eg., 10 steps) for the continuous control of stroke size during testing. But for the model in [JLY*18], it should use 10 different residual units to achieve comparable results, which seems

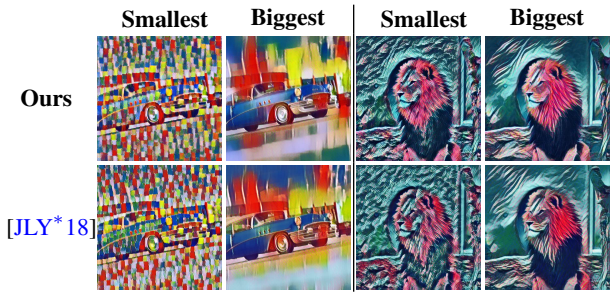


Figure 8. Visual comparisons of producing a wide span of stroke sizes.

extremely wasteful and liable to overfitting. Besides, during testing, the computation graph of our model can be adjusted flexibly depending on the actual demand by only changing the recurrent time while for [JLY*18], it seems a little troublesome to do this, especially on the tensorflow platform. Figure 8 shows several results. While achieving comparable results, ours shows less high-frequency noise and saves a lot of parameters, making it easier to train and more suitable for mobile development.

4.3. Multi-styles Stroke-controllable Transfer

In this section, we embed our *recurrent stroke-pyramid* into the Multi-Styles Model. Clearly, we use CIN [DSK16] as our S-N method to transfer multiple styles while the stroke size for each style could be regulated by the *recurrent stroke-pyramid*. We call our proposed model using IN as 1-style model while that using CIN as N-styles model, where N indicates the number of styles. We first train 10 1-style models and 1 10-styles model, then compare between them. Finally, we execute two run-time control strategies described in Section 3.2, further manifesting the flexibility of it.

4.3.1. Comparison

To validate the effectiveness of the embedding method, we randomly sample 5 styles and compare the final style and content losses for each stroke size between 1-style models and 10-styles model (left column of Figure 9). The losses are all averaged over 32 random batches of content images. Generally speaking, for style loss the 10-styles model achieves around $1.43\% \pm 5.02\%$ lower losses than its counterparts while for content loss, the difference is more negligible (about $0.64\% \pm 1.37\%$ higher). Since our N-styles model uses only one model to transfer multiple styles and stroke sizes, this difference can be ignored in the view of much parameters reduction compared to N 1-style models. The right column of Figure 9 shows several visual comparisons between N-styles model and N 1-style models. We can see that both results are qualitatively similar.

4.3.2. Run-time Control

Stratregy 1. Concurrently Interpolating Between Styles and Stroke Sizes. Similar to [DSK16], interpolating between different styles' transformation parameters W_s and μ_s can fuse them together. During style fusing, if we use the Function 2 of our fusion module detailed in Section 3.2, then the stroke size of the fused style can

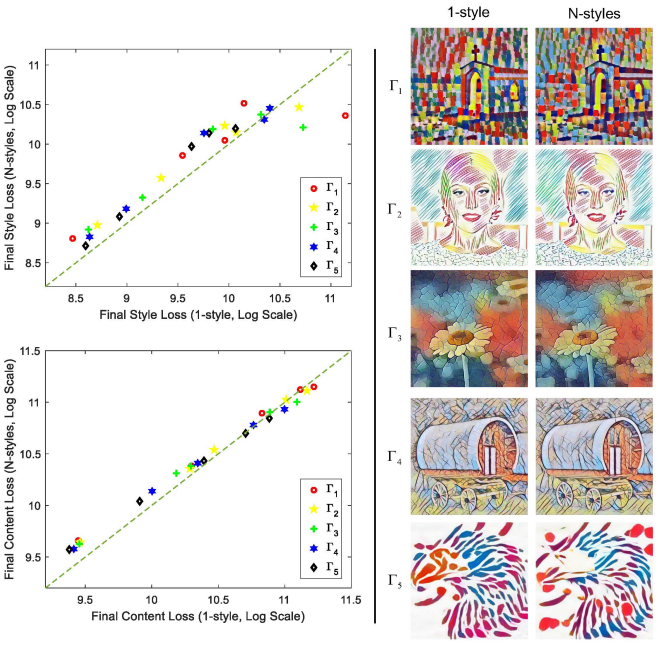


Figure 9. The left column is the final average loss comparisons for each stroke size between 1-style models and N-styles model. Different shapes represent different stroke sizes. The right column is the visual comparisons between them.

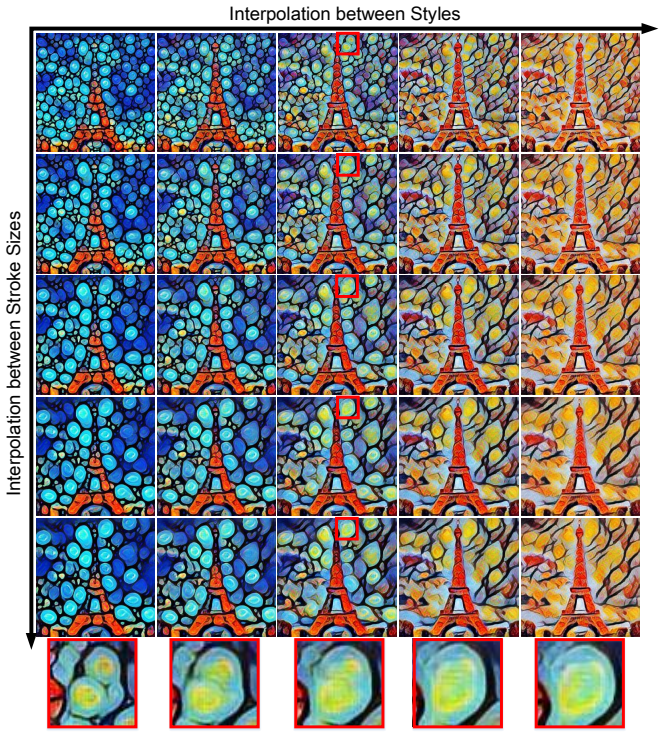


Figure 10. Interpolation between both styles and stroke sizes.

also be continuously controlled, as shown in Figure 10. It can be seen that the stroke size is changed faily smoothly. While our model is only trained to control the stroke size for the style chosen from the fixed style set, it generalizes naturally to the fused style.

Stratregy 2. Concurrently Executing Style and Stroke Size Spatial Control. Since the feature statistics in the generator can reflect the style of the output image, applying CIN separately to different regions via region masks can achieve style spatial control. The masks for different layers can be obtained by using nearest neighbor interpolation according to current layer's stride. By concurrently using the Function 3 of our fusion module detailed in Section 3.2, where the masks are obtained by the same method, ours can achieve both stroke size and style spatial control in a completely feed-forward manner. Figure 11 shows some results.

4.4. Arbitrary-style Stroke-controllable Transfer

According to Section 4.3, our *recurrent stroke-pyramid* can successfully control the stroke size for each style in the fixed style set. Thus, it's natural to combine the arbitrary style transfer architecture with it to flexibly control the stroke size of arbitrary style without any image pre-processing. Here, we employ the pre-trained network proposed by [GLK*17] and simply add *our recurrent stroke-pyramid* after its fifth residual unit. The *recurrent stroke-pyramid* is also followed by a fusion module to control the stroke size of the output image. It should be noted that W_s and μ_s in the *recurrent stroke-pyramid* are also computed from the pre-trained Inception-v3 [SVI*16], just like the way in [GLK*17]. During training, we use our proposed training strategy to first train the newly-added parameters with others fixed and then fine-tune the whole network, on the DTD dataset [CMK*14]. Figure 1 shows some results (image (e) and (f)). More results can be found in the Figure 12. It can be seen that the network generalizes well to the unseen styles.

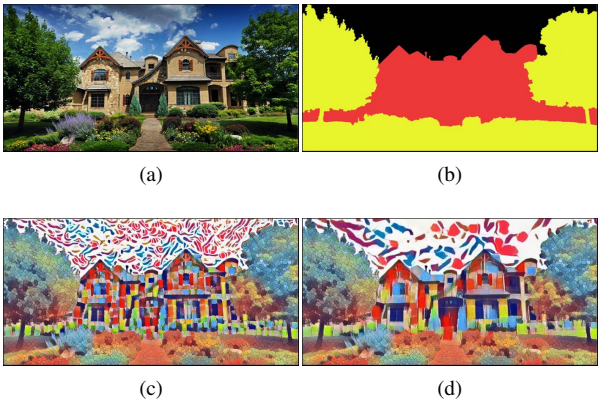


Figure 11. **(a)** Content image. **(b)** The mask for style and stroke size spatial control. The yellow region corresponds to the style image (e) of Figure 14 in the Appendix and stroke size Γ_1, the red region the style image (b) and stroke size Γ_3, the black region the style image (d) and stroke size Γ_5. **(c)** The stylized result without stroke size spatial control. **(d)** The stylized result with stroke size spatial control.

Figure 12. The samples of arbitrary-style stroke-controllable transfer. The model generalizes well to the unseen styles, especially the styles with regular patterns (see the first, third and fifth rows).

5. Conclusion

In this work, we propose a recurrent convolutional neural subnetwork to control the stroke size of the generated images in Fast Style Transfer, which is a brand-new concept and also a novel technique in the style transfer literature. It can automatically increase the receptive field of the whole network without introducing extra parameters, making it flexible to generate larger unseen stroke size than during training. What's more, it can save a lot of memories when a wide range of stroke sizes is needed. Experimental results demonstrate that ours can not only achieve comparable results with much fewer parameters but allow more flexibility and efficiency, compared with the original method. In addition, the proposed method can be successfully extended to Multi-Styles model or even Arbitrary-Style model to control both the style and the stroke size in an entirely feed-forward manner. However, the combinational optimization over multiple styles and stroke sizes will compromise the rendering quality (e.g., row 6 and 7 in Figure 12). Thus, further research is needed to generate high-quality images for arbitrary-style stroke-controllable transfer.

Acknowledgments

This work is partially supported by the National Science Foundation of China under Grant no. 61502306, 61601112, the Fundamental Research Funds for the Central Universities, DHU Distinguished Young Professor Program and the China Young 1000 Talents Program.

References

[CMK*14] CIMPOI M., MAJI S., KOKKINOS I., MOHAMED S., VEDALDI A.: Describing textures in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on* (2014), IEEE, pp. 3606–3613. 8

[CYL*17] CHEN D., YUAN L., LIAO J., YU N., HUA G.: Stylebank: An explicit representation for neural image style transfer. In *Proc. CVPR* (2017). 1

[DSK16] DUMOULIN V., SHLENS J., KUDLUR M.: A learned representation for artistic style. *CoRR, abs1610.07629 2*, 4 (2016), 5. 1, 2, 3, 4, 7

[GEB15a] GATYS L., ECKER A. S., BETHGE M.: Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems* (2015), pp. 262–270. 1, 4

[GEB15b] GATYS L. A., ECKER A. S., BETHGE M.: A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576* (2015). 1, 4

[GEB*17] GATYS L. A., ECKER A. S., BETHGE M., HERTZMANN A., SHECHTMAN E.: Controlling perceptual factors in neural style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). 1

[GG01] GOOCH B., GOOCH A.: *Non-photorealistic rendering*. AK Peters/CRC Press, 2001. 1

[GLK*17] GHIASI G., LEE H., KUDLUR M., DUMOULIN V., SHLENS J.: Exploring the structure of a real-time, arbitrary neural artistic stylization network. *arXiv preprint arXiv:1705.06830* (2017). 1, 2, 3, 4, 5, 6, 8

[HB17] HUANG X., BELONGIE S.: Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR, abs1703.06868* (2017). 1, 2, 3, 4

[HZRS16a] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778. 3

[HZRS16b] HE K., ZHANG X., REN S., SUN J.: Identity mappings in deep residual networks. In *European Conference on Computer Vision* (2016), Springer, pp. 630–645. 3

[IS15] IOFFE S., SZEGEDY C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015). 3

[JAFF16] JOHNSON J., ALAHI A., FEI-FEI L.: Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision* (2016), Springer, pp. 694–711. 1, 3, 4, 5, 6

[JLY*18] JING Y., LIU Y., YANG Y., FENG Z., YU Y., SONG M.: Stroke controllable fast style transfer with adaptive receptive fields. *arXiv preprint arXiv:1802.07101* (2018). 1, 2, 3, 4, 5, 6, 7

[Jul81] JULESZ B.: Textons, the elements of texture perception, and their interactions. *Nature 290*, 5802 (1981), 91. 3

[KA15] KINGA D., ADAM J. B.: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)* (2015). 5

[KEBK05] KWATRA V., ESSA I., BOBICK A., KWATRA N.: Texture optimization for example-based synthesis. In *ACM Transactions on Graphics (ToG)* (2005), vol. 24, ACM, pp. 795–802. 1

[LFY*17a] LI Y., FANG C., YANG J., WANG Z., LU X., YANG M.-H.: Diversified texture synthesis with feed-forward networks. In *Proc. CVPR* (2017). 1

[LFY*17b] LI Y., FANG C., YANG J., WANG Z., LU X., YANG M.-H.: Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems* (2017), pp. 385–395. 1, 2, 3, 5, 6

[LMB*14] LIN T.-Y., MAIRE M., BELONGIE S., HAYS J., PERONA P., RAMANAN D., DOLLÁR P., ZITNICK C. L.: Microsoft coco: Common objects in context. In *European conference on computer vision* (2014), Springer, pp. 740–755. 5

[LW16] LI C., WAND M.: Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European Conference on Computer Vision* (2016), Springer, pp. 702–716. 1

[SLSW18] SHENG L., LIN Z., SHAO J., WANG X.: Avatar-net: Multi-scale zero-shot style transfer by feature decoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 8242–8250. 1, 2, 3, 5, 6

[SS02] STROTHOTTE T., SCHLECHTWEG S.: *Non-photorealistic computer graphics: modeling, rendering, and animation*. Morgan Kaufmann, 2002. 1

[SVI*16] SZEGEDY C., VANHOUCKE V., IOFFE S., SHLENS J., WOJNA Z.: Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2818–2826. 3, 8

[SZ14] SIMONYAN K., ZISSERMAN A.: Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014). 4, 5

[ULVL16] ULYANOV D., LEBEDEV V., VEDALDI A., LEMPITSKY V. S.: Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML* (2016), pp. 1349–1357. 1, 3, 4

[UVL16] ULYANOV D., VEDALDI A., LEMPITSKY V. S.: Instance normalization: The missing ingredient for fast stylization. *CoRR abs/1607.08022* (2016). URL: http:\/\/arxiv.org\/abs\/1607.08022, arXiv:1607.08022. 2

[UVL17] ULYANOV D., VEDALDI A., LEMPITSKY V.: Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proc. CVPR* (2017). 1, 2

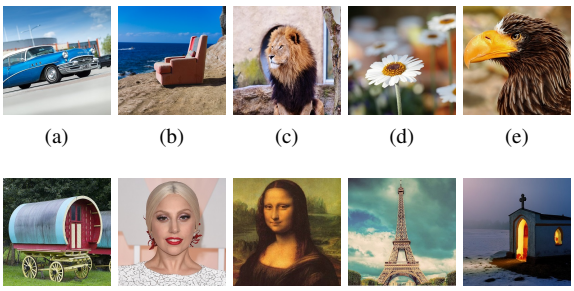
[WOZW17] WANG X., OXHOLM G., ZHANG D., WANG Y.-F.: Multimodal transfer: A hierarchical deep convolutional neural network for fast artistic style transfer. *arXiv preprint* (2017). 3

[ZD17] ZHANG H., DANA K.: Multi-style generative network for real-time transfer. *arXiv preprint arXiv:1703.06953* (2017). 1

[ZGWX05] ZHU S.-C., GUO C.-E., WANG Y., XU Z.: What are textons? *International Journal of Computer Vision 62*, 1-2 (2005), 121–143. 3

6. Appendix

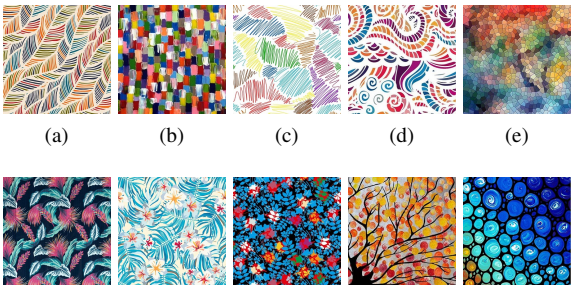
Figure 13 and Figure 14 show the content images and style images respectively used in Section 4.2 and 4.3. Algorithm 1 details our training strategy.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 13. Diverse content images used in 4.2 and 4.3.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 14. Diverse style images used in 4.2 and 4.3.

Algorithm 1 Our Training Strategy

for number of training iterations **do**
  Sample a style image I_s from $\{I_s\}$
  $w \leftarrow (w_d, w_\theta)$, $w_\theta \leftarrow \mathcal{N}_{meta}(I_s)$
  for i *in* $\{1, 2, ..., R\}$ **do**
    $I_s^{\Gamma_i} \leftarrow \text{Resize}(I_s; \Gamma_i)$
    Sample a mini-batch of content: $\{(I_c)^j\}_{j=1}^m$
    $\{(I^{\Gamma_i})^j\}_{j=1}^m \leftarrow \mathcal{N}_f(\{(I_c)^j\}_{j=1}^m; w, \Gamma_i)$
    Compute gradients and update w.
  end for
end for