Stylized Neural Painting

Zhengxia Zou[1], Tianyang Shi[2], Shuang Qiu[1], Yi Yuan[2], Zhenwei Shi[3]

[1]University of Michigan, Ann Arbor, [2]NetEase Fuxi AI Lab, [3]Beihang University

Photo to oil painting Photo to oil painting Photo to marker-pen painting Photo to watercolor painting

Input photo Output oil painting Stylized output (transfer color only) Input photo Output oil painting Stylized output (transfer color & texture)

Figure 1: We propose a stroke-rendering based method that generates realistic painting artworks. The paintings are generated with a vector format, which can be further optimized and rendered with different visual styles. We recommend zooming in to see the stroke textures.

Abstract

This paper proposes an image-to-painting translation method that generates vivid and realistic painting artworks with controllable styles. Different from previous image-to-image translation methods that formulate the translation as pixel-wise prediction, we deal with such an artistic creation process in a vectorized environment and produce a sequence of physically meaningful stroke parameters that can be further used for rendering. Since a typical vector render is not differentiable, we design a novel neural renderer which imitates the behavior of the vector renderer and then frame the stroke prediction as a parameter searching process that maximizes the similarity between the input and the rendering output. We explored the zero-gradient problem on parameter searching and propose to solve this problem from an optimal transportation perspective. We also show that previous neural renderers have a parameter coupling problem and we re-design the rendering network with a rasterization network and a shading network that better handles the disentanglement of shape and color. Experiments show that the paintings generated by our method have a high degree of fidelity in both global appearance and local textures. Our method can be also jointly optimized with neural style transfer that further transfers visual style from other images. Our code and animated results are available at https://jiupinjia.github.io/neuralpainter/.

1. Introduction

Creating artistic paintings is one of the defining characteristics of humans and other intelligent species. In recent years, we saw great advancements in generative modeling of image translation or style transfer which utilizes neural network as a generative tool [6, 13, 23, 38]. Previous image-to-image translation and style transfer methods typ-

ically formulate the translation either as a pixel-wise mapping [13, 38] or a continuous optimization process in their pixel space [6]. However, as an artistic creation process, the paintings usually proceed as a sequentially instantiated process that creates using brushes, from abstract to concrete, and from macro to detail. This process is fundamentally different from how neural networks create artwork that produces pixel-by-pixel results. To fully master the professional painting skills, people usually need a lot of practice and learn domain expertise. Even for a skilled painter with years of practice, it could still take hours or days to create a realistic painting artwork.

In this paper, we explore the secret nature of human painting and propose an automatic image-to-painting translation method that generates vivid and realistic paintings with controllable styles. We refer to our method as "Stylized Neural Painter". Instead of manipulating each of the pixels in the output image, we simulate human painting behavior and generate vectorized strokes sequentially with a clear physical significance. Those generated stroke vectors can be further used for rendering with arbitrary output resolution. Our method can "draw" in a variety of painting styles, e.g. oil-painting brush, watercolor ink, marker-pen, and tape art. Besides, our method can also be naturally embedded in a neural style transfer framework and can be jointly optimized to transfer its visual style based on different style reference images.

In our method, different from the previous stroke-based rendering methods that utilize step-wise greed search [8, 19], recurrent neural network [7], or reinforcement learning [5, 11, 33, 36], we reformulate the stroke prediction as a "parameter searching" process that aims to maximize the similarity between the input and the rendering output in a self-supervised manner. Considering that a typical graphic renderer is not differentiable, we take advantage of the neural rendering that imitates the behavior of the graphic rendering and make all components in our method differentiable. We show that previous neural stroke renderers [11, 29, 30] may suffer from the parameter coupling problem when facing complex rendering scenarios, e.g., brushes with real-world textures and color-transition. We, therefore, re-design the neural renderer and decomposed the rendering architecture into a rasterization network and a shading network, which can be jointly trained and rendered with much better shape and color fidelity. We also found interestingly that the pixel-wise similarity like ℓ_1 or ℓ_2 pixel loss, may have an intrinsic flaw of zero-gradient on optimizing over the vectorized parameters, although these losses have been widely used in a variety of image translation tasks [13, 17, 38]. We show that this problem lies in the different nature of stroke parameterization and rasterization, and propose to solve this problem from the perspective of optimal transportation. Specifically, we consider the movement of a stroke from one location to another as a transportation process, where we aim to minimize the efforts of that movement.

We test our method on various real-world images and photos, including human portraits, animals, scenery, daily objects, art photography, and cartoon images. We show that our method can generate vivid paintings with a high degree of realism and artistic sense in terms of both global visual appearance and local texture fidelity.

The contribution of our paper is summarized as follows:

- We propose a new method for stroke based image-to-painting translation. We re-frame the stroke prediction as a parameter searching processing. Our method can be jointly optimized with neural style transfer in the same framework.

- We explore the zero-gradient problem on parameter searching and view the stroke optimization from an optimal transport perspective. We introduce a differentiable transportation loss and improves stroke convergence as well as the painting results.

- We design a new neural renderer architecture with a dual-pathway rendering pipeline (rasterization + shading). The proposed renderer better deals with the disentanglement of the shape and color and outperforms previous neural renderers with a large margin.

2. Related Work

Image translation and style transfer. Image translation, which aims at translating images from one domain (e.g., real photos) to another (e.g., artworks), has drawn great attention in recent years. GAN based image translation such as Pix2Pix [12], CycleGANs [38] and their variants [26] has played an important role in tasks like image synthesis, semantic editing, style transfer, and also has been applied to computer-generated arts [1]. In addition to the GAN based method, neural style transfer has also made breakthroughs in stylized image synthesis and is widely used for artwork creation [6, 13]. Besides, the Deep Dream [23], a method that was initially designed to help visualize deep neural networks, also has become a new form of psychedelic and abstract art. Despite the above applications, these methods all generate paintings in a pixel-by-pixel manner, which deviates from the fact that humans use brushes to paint.

Differentiable rendering. Rendering is a fundamental problem in computer graphics that converts 3D models into 2D images. Traditional rendering pipelines typically involve a discrete operation called rasterization, which makes the rendering non-differentiable. Differentiable rendering [14, 18, 20, 21] breaks such limitations and allows calculation of the derivative from the rendering output to
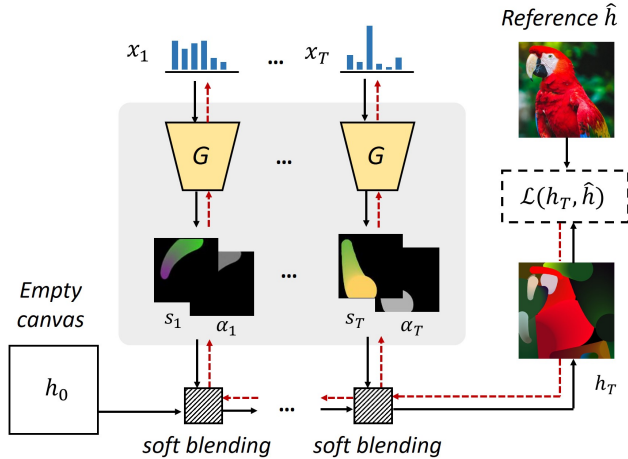
Figure 2: We start from an empty canvas and then render stroke-by-stroke with soft blending. We use gradient descent to find a set of "optimal" stroke parameters that minimize the loss $\mathcal{L}$. Here black arrowlines mean forward propagation and red ones mean back-propagation of the gradient.

the input parameters such as shape, camera pose, and lighting. Since deep neural networks are naturally differentiable in their topology, a new research topic called "neural rendering" quickly emerged [4, 25, 29], which bridges the gap between graphic rendering and deep neural networks.

Image sketching/painting. Humans are born with the ability to abstract and sketch object instances. Early methods on image sketching deal with this problem by using step-wise greedy search or require user interaction [9]. Recent approaches typically train recurrent neural networks [7] and Reinforcement Learning (RL) agent [5, 33, 36], or integrate adversarial training [24] to generate nondeterministic stroke sequences. More recently, thanks to the recent advances of neural rendering [4, 25], computers are now able to generate more realistic painting artworks [5, 11, 24]. Among these methods, "Learning to Paint" [36] has a similar research motivation to ours, both dedicated to generating stroke based realistic paintings. However, our method differs from this method in several aspects. First, this method generates strokes by using RL while we formulate this process as stroke parameter searching since training RL agents is computationally expensive. Second, we focus on stroke-based style transfer, which is a rare studied problem. Finally, also we redesign the neural renderer and introduce optimal transport methods to this problem.

3. Methodology

Our method consists of three technical modules: 1) a neural renderer that is trained to generate strokes given a

set of vectorized stroke parameters; 2) a stroke blender that combines multiple rendering strokes in a differentiable manner; 3) a similarity measurement module that enforces the reconstruction of the input image. In the following, we introduce how each module works accordingly.

3.1. Overview

Fig. 2 shows an overview of our method. Given an empty canvas h_0, we draw step-by-step and superimpose those strokes rendered at each step iteratively. In each drawing step t, a trained neural renderer G takes in a set of stroke parameters $\boldsymbol{x}_t$ (e.g., shape, color, transparency, and texture), and produces a stroke foreground s_t and an alpha matte α_t. We then use soft blending to mix the canvas, the foreground, and alpha matte at each step t and make sure the entire rendering pipeline is differentiable. The soft blending is defined as follows:

$$h_{t+1} = \alpha_t s_t + (1 - \alpha_t)h_t, \tag{1}$$

where $(s_t, \alpha_t) = G(\boldsymbol{x}_t)$. We finally gather the stroke parameters from all the T steps and optimize them by searching within the stroke parameter space. The searching is conducted under a self-supervised manner, i.e., we enforce the final rendered output h_T similar to a reference image $\hat{h}$:

$$h_T = f_{t=1 \sim T}(\tilde{\boldsymbol{x}}) \approx \hat{h}, \tag{2}$$

where $f_{t=1 \sim T}(\cdot)$ is a recursive mapping from stroke parameters to the rendered canvas. $\tilde{\boldsymbol{x}} = [\boldsymbol{x_1}, ..., \boldsymbol{x}_T]$ are the collection of stroke parameters at $t = 1, 2, ...T$ drawing steps.

Suppose $\mathcal{L}$ represents a loss function that measures the similarity between the canvas h_T and the reference $\hat{h}$, we optimize all the input strokes $\tilde{\boldsymbol{x}}$ at their parameter space and minimize the facial similarity loss $\mathcal{L}$: $\tilde{\boldsymbol{x}}^\star = \arg\min_{\tilde{\boldsymbol{x}}} \mathcal{L}(h_T, \hat{h})$. We use gradient decent to update the strokes as follows:

$$\tilde{\boldsymbol{x}} \leftarrow \tilde{\boldsymbol{x}} - \mu \frac{\partial \mathcal{L}(h_T, \hat{h})}{\partial \tilde{\boldsymbol{x}}}, \tag{3}$$

where μ is a predefined learning rate.

3.2. Disentangle neural rendering

To build a neural renderer, a general practice is to build a deep convolutional network and train it to imitate the behavior of a graphic engine. Previous researches proposed to use stacked transposed convolutions [29, 37] or position encoder + decoder architecture [11, 35]. These approaches can work well in simple stroke rendering scenarios. However, we find these renderers may suffer from a coupling of shape and color representations when testing with more complex rendering settings like color transition and stroke textures. We propose to solve this problem by designing
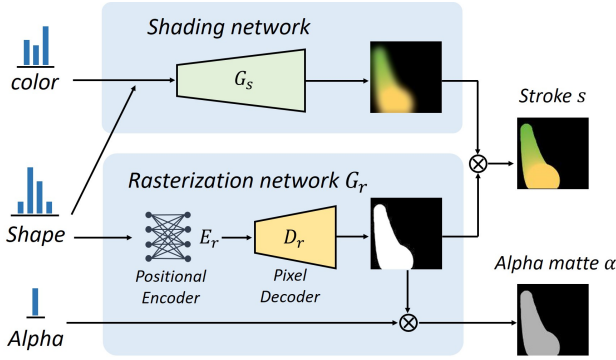
Figure 3: We design a dual-pathway neural renderer which consists of a shading network G_s and a rasterization network G_r. Our renderer takes in a group of stroke parameters (color, shape, and transparency) and produces the rasterized foreground map and alpha matte.

a dual-pathway neural renderer that disentangles color and shape/texture through the rendering pipeline.

As shown in Fig. 3, the proposed neural renderer consists of two networks, a shading network G_s and a rasterization network G_r. We divide the parameters of a stroke $\boldsymbol{x}$ into three groups: color, shape, and transparency. We build G_s as a stack of several transposed convolution layers [34], which takes in both the color and shape parameters and generates strokes with faithful foreground color. We design the G_r as a positional encoder + a pixel decoder, which simply ignores the color but generates stroke silhouette with a clear shape boundary. We finally generate the output stroke foreground s by masking the color map with the stroke silhouette and generate the final alpha matte α by rescaling the silhouette using the input alpha value.

We train our neural renderer with standard ℓ_2 pixel regression losses on both the rendered stroke foreground and the alpha matte. During the training, we minimize the following objective function:

$$\mathcal{L}_G(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{x} \sim u(\boldsymbol{x})} \{\|s - \hat{s}\|_2^2 + \|\alpha - \hat{\alpha}\|_2^2\}, \qquad (4)$$

where $\hat{s}$ and $\hat{\alpha}$ are the ground truth foreground and alpha matte rendered by the graphic engine. $\boldsymbol{x} \sim u(\boldsymbol{x})$ are stroke parameters randomly sampled within their parameter space.

3.3. Pixel similarity and zero-gradient problem

There are many ways to define the similarity between the rendered output h_T and the reference $\hat{h}$, and perhaps the most straight-forward one is to define as pixel-wise loss, e.g., ℓ_1 or ℓ_2 losses. Note that when we manipulate the image by directly optimizing from their pixel space, using the pixel-wise loss can work pretty well. However, when it comes to optimizing stroke parameters, we show that pixel loss does not always guarantee effective gradient de-
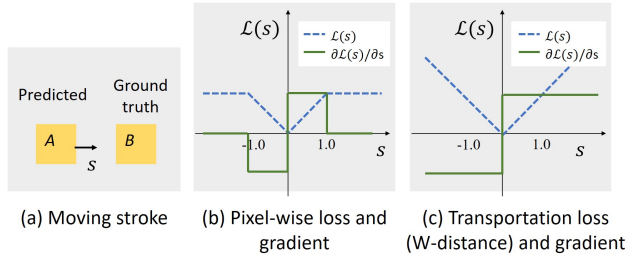

(a) Moving stroke (b) Pixel-wise loss and gradient (c) Transportation loss (W-distance) and gradient

Figure 4: A simple example to explain why pixel-wise loss $\mathcal{L}$ may have the zero-gradient problem ($\partial \mathcal{L}/\partial s = 0$) during the stroke update. (a) When we move a square-shaped stroke A along the s direction to its target B, the pixel loss would remain constant when there is no overlap between A and B. This will cause zero-gradients in the loss function, as shown in (b). As a comparison, the optimal transportation (OT) loss in (c) does not have such a problem since the loss is correlated to the physical distance between A and B.

scent. Particularly, when the rendered stroke and its ground truth do not share overlapped regions, there will be a zero-gradient problem. In Fig. 4, we give a simple example of why such cases would happen.
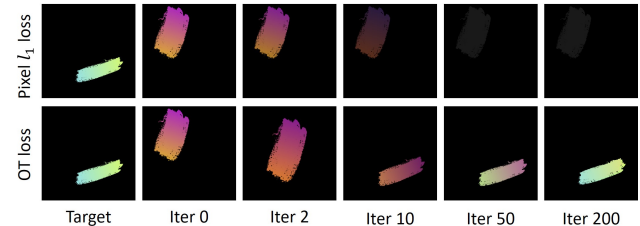

Figure 5: A comparison between the pixel loss (1st row) and transportation loss (2nd row) on "pushing" a stroke from its initial state to its target. Using the proposed transportation loss, the stroke nicely converged to the target. As a comparison, the pixel loss fails to converge due to the zero-gradient problem in its position and scale.

Suppose we have a square-shaped stroke A generated by a renderer and we aim at optimizing its parameters to make it converge to the target location B. If we move A along the horizontal direction and we suppose this movement is controlled by a certain parameter dimension (e.g., center x-coordinate). Clearly, when there is no overlap between the A and B, the sum of the pixel loss between the canvas and the ground truth will remain a constant, which leads to a zero gradient and fails to guide the stroke update. Fig. 4 (b) shows the pixel loss and its gradient on the moving distance. This problem can be effectively solved by re-defining the loss as the amount of transportation effort spent on the movement, where the farther A and B are, the greater the

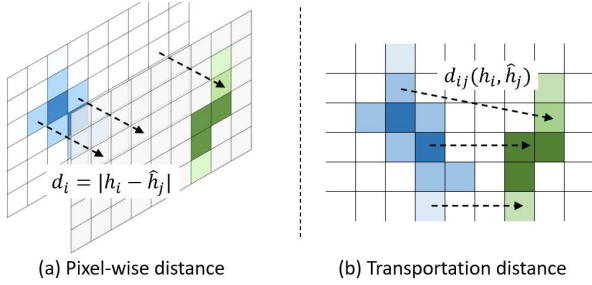(a) Pixel-wise distance   (b) Transportation distance

Figure 6: A comparison between the pixel-wise distance and the transportation distance.

effort required, as shown in Fig. 4 (c). In Fig. 5, we give another visualization example of how strokes behave when optimizing the pixel-loss and the transportation loss. We see that with pixel ℓ_1 loss, the stroke fails to move along the right direction since there is no gradient on its parameterized locations while using transportation loss makes the stroke nicely converges to the target.

3.4. Optimal transport for stroke searching

We define the minimum transportation efforts, i.e., the Wasserstein distance, as an effective measure of similarity loss between the canvas and the reference image. Fig. 12 gives a brief visualization of how optimal transport loss differs from pixel loss on measuring the image similarity.

Given a rendered canvas h and a reference image $\hat{h}$, we define their normalized pixel values p_h and $\hat{p}_h$ as their probabilistic marginal functions. Here we omit the subscript T for simplicity. We define $\mathbf{P} \in \mathbb{R}_+^{n \times n}$ as the joint probability matrix whose (i, j)-th element denotes the joint probability of the i-th pixel in h and j-th pixel in $\hat{h}$, where n is the number of pixels in the image. We let $\mathbf{D}$ be the cost matrix whose (i, j)-th element denotes the Euclidean distance between the i-th pixel's location in h and j-th pixel's location in $\hat{h}$. Thus, the matrix $\mathbf{D}$ list all the labor costs of moving a "unit pixel mass" from one position in h to another one in $\hat{h}$. In the discrete case, the classic optimal transport distance can be written as a linear optimization problem $\min_{\mathbf{P} \in \mathcal{U}} \langle \mathbf{D}, \mathbf{P} \rangle$, where $\mathcal{U} := \{\mathbf{P} \in \mathbb{R}_+^{n \times n} \mid \mathbf{P} \mathbf{1}_n = p_h, \mathbf{P}^T \mathbf{1}_n = \hat{p}_h\}$.

In this paper, we employ a smoothed version of the classic optimal transport distance with an entropic regularization term, which yields the celebrated Sinkhorn distance [3]. The Sinkhorn distance is differentiable [22] and enjoys benign mathematical properties that can result in much lower computational cost than the original one [3]. The primary idea is to consider an extra entropic constraints on the joint probability matrix $\mathbf{P}$ apart from $\mathcal{U}$. Further using the Lagrange multiplier, one can transform the problem into a regularized form as (5). We let $\mathcal{L}_{ot}$ be the optimal trans-

portation loss, namely the minimum transportation effort on moving strokes from one location to another, and then define it as the Sinkhorn distance in the following way

$$
\begin{aligned}
&\mathcal{L}_{ot}(h, \hat{h}) := \langle \mathbf{D}, \widetilde{\mathbf{P}}_\lambda \rangle, \text{ with} \\
&\widetilde{\mathbf{P}}_\lambda = \underset{\mathbf{P} \in \mathcal{U}}{\operatorname{argmin}} \ \langle \mathbf{D}, \mathbf{P} \rangle - \frac{1}{\lambda} E(\mathbf{P}),
\end{aligned} \tag{5}
$$

where the entropy $E(\mathbf{P}) := -\sum_{i,j=1}^n \mathbf{P}_{i,j} \log \mathbf{P}_{i,j}$.

The optimized transport loss can thus be easily integrated to the parameter searching pipeline and can be easily optimized together with other losses. We finally define the similarity loss as a combination of pixel ℓ_1 loss and the optimal transportation loss:

$$
\mathcal{L} = \beta_{\ell_1} \mathcal{L}_{\ell_1} + \beta_{ot} \mathcal{L}_{ot}, \tag{6}
$$

where β's control the balance of the above two objectives.

3.5. Joint optimization with neural style transfer

Since we frame our stroke prediction under a parameter searching paradigm, our method naturally fits the neural style transfer framework. Neural style transfer models are typically designed to updating image pixels to minimize the combination of a content loss and a style loss [6] $\mathcal{L}_{nst} = \mathcal{L}_{ctt} + \mathcal{L}_{sty}$, where $\mathcal{L}_{ctt}$ and $\mathcal{L}_{sty}$ correspond to the constraints on the image content and style respectively.

To edit the global visual style of the rendering output, we extend the similarity loss (6) with the style loss above, and define the extended similarity loss as follows:

$$
\mathcal{L} = \beta_{\ell_1} \mathcal{L}_{\ell_1} + \beta_{ot} \mathcal{L}_{ot} + \beta_{sty} \mathcal{L}_{sty}, \tag{7}
$$

where β's are the weights to balance the losses. We follow Gatys *et al.* [6] and compute the style loss as the square error of the Gram matrices of the features extracted by deep CNNs. We use VGG-19 [32] as the feature extraction network and compute the Gram matrices on the features produced by the 2nd, 4th, and 7th convolutional layers.

3.6. Implementation Details

Network architecture. We build our shading network similar to DCGAN's generator [27], which consists of six transposed conv layers. We remove the Tanh activation from the output layer and observe a better convergence. In our rasterization network, we follow Huang *et al.* [11] and first build a positional encoder with four fully-connected layers then build a pixel decoder with six conv layers and three pixel-shuffle layers [31]. We also experiment with the architecture of "UNet" [28], wherein this case, we tile the stroke parameters on their spatial dimensions to a 3D tensor as the input of the network.

Training details. We train our renderer by using Adam optimizer [15]. We set batch size to 64, learning rate to 2e-4,
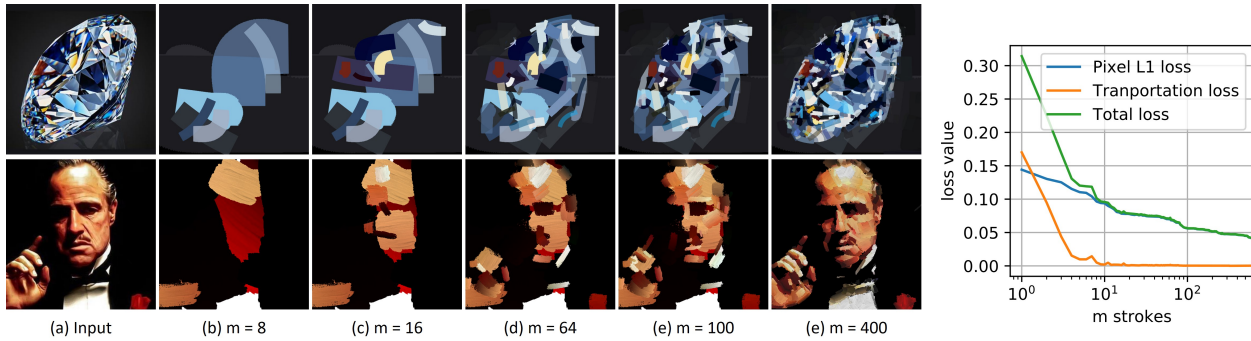
Figure 7: Stroke-by-stoke painting result of our method with marker-pen (1st row) and oil-paint brush (2nd row). On the right, we also plot the loss curves (pixel ℓ_1 loss, transportation loss, and total loss) as the painting proceeded.
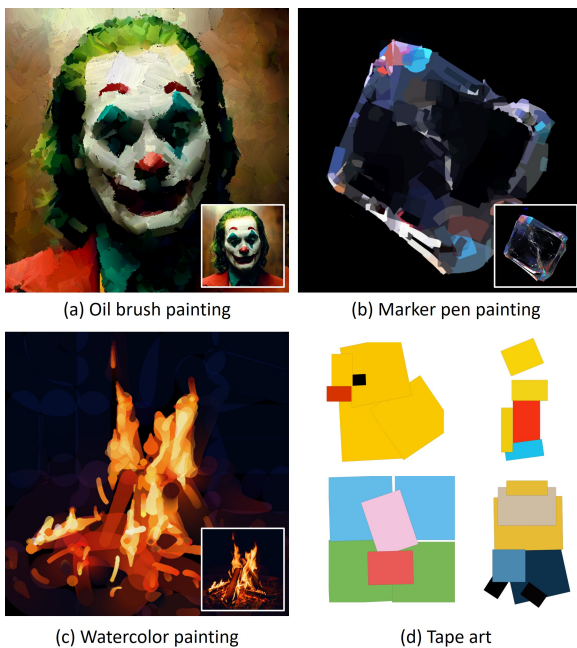

Figure 8: (a)-(c) Stylized paintings generated by our method. In (d), we also show some highly abstract tape arts of cartoon characters generated by our method. Can you guess who they are? (See answers below)

and betas to (0.9, 0.999). We reduce the learning rate to its 1/10 every 100 epochs and stop training after 400 epochs. In each epoch, we randomly generate 50,000x64 ground truth strokes using a vector engine. We set the rendering output size to 128x128 pixels. We train renderers separately for each stroke type.

Progressive rendering. To render with more details, we design a progressive rendering pipeline in both the scale and action dimension. We first start from searching parameters on a single 128x128 canvas and then divide the canvas into $m \times m$ blocks ($m = 2, 3, 4, ...$) with overlaps and

search on each of them accordingly. In each block scale, we gradually add new strokes to an "active set" and update the strokes progressively. In each update, we optimize all strokes within the active set at the same time. We run gradient descent for $20 \times N$ steps for each block, where N is the number of strokes in each block, same for all blocks regardless of their scales.

Other details. We set $\beta_{\ell_1} = 1.0$ and $\beta_{ot} = 0.1$. We use the RMSprop [10] optimizer for gradient descent with learning $\mu = 0.01$. In the style transfer, we set $\beta_{sty} = 0.5$ and update for 200 steps. In Sinkhorn loss, we resize the canvas to 48x48 pixels to speed up the computation. We set the entropic regularization term $\epsilon = 0.01$, and the number of steps in sinkhorn loop $n_{iter} = 5$. For more details on the network architectures and stroke parameterization, please refer to our supplementary material.

4. Experimental Analysis

4.1. Stylized painting generation

Fig. 8 shows a group of stylized paintings generated by using our method with different stroke types. In (d), we also show several tape-artworks of well-known characters that are automatically created by our method with a minimum number of strokes. We can see that our method successfully learns high-level abstractions of the characters and vividly portrays their shape and color. Fig. 7 shows the stroke-by-stroke painting results by using different stroke brushes. On the right of this figure, we also plot the changes in the loss values as the painting proceeded. We can see that in the very first few drawing steps, our method nicely captures the global appearance of the object, and the drawing then gradually goes from macro to detail. Fig. 9 shows more examples of our painting results as well as their style transfer results. We can see by integrating the style loss in (7), both color and textures can be successfully transferred to the paintings with their content remaining unchanged.

---

[0] Answer: Rubber Duck, Bart Simpson, Peppa Pig, and, Minions.

Image to painting   Stylized outputs (transfer both color & texture)   Image to painting   Stylized outputs (transfer both color & texture)

Image to painting   Stylized outputs (transfer color only)   Image to painting   Stylized outputs (transfer color only)
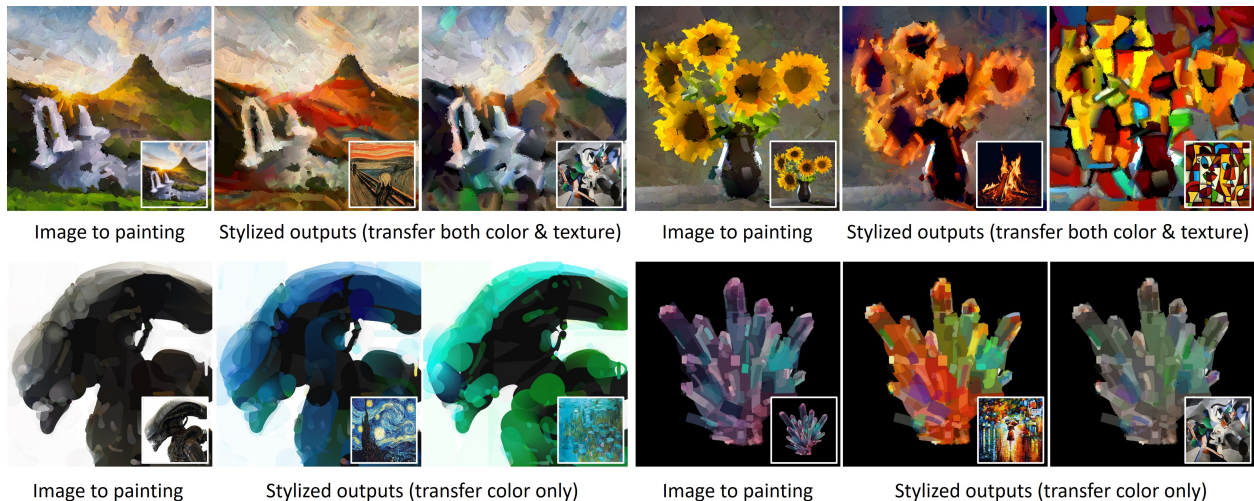
Figure 9: Style transfer results by using our method. The 1st row shows the results that transfer the style of both color and texture. The 2nd row shows the results that transfer color only.
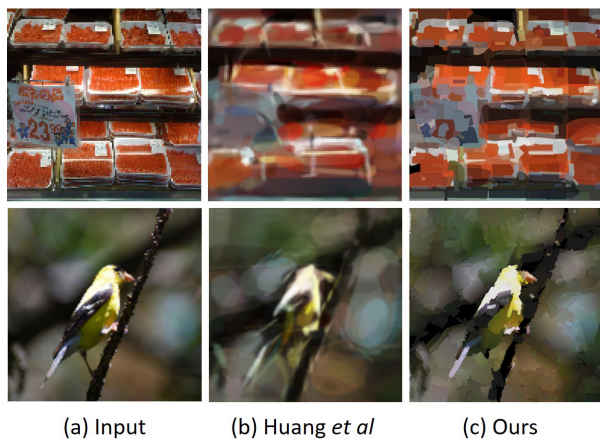

(a) Input   (b) Huang *et al*   (c) Ours

Figure 10: A comparison of the paintings created by our method and "Learning-to-Paint" [11] (400 strokes).

(a) Input   (b) Manual art by A. Lister   (c) Auto-creation (ours)

Figure 11: Artworks created manually by a New York artist Adam Lister and created automatically by our method.

4.2. Comparison with other methods

In Fig. 10, we compare our method with a recent proposed stroke-based image-to-painting translation method "Learning-to-Paint" [11]. This method trains an RL agent to generate painting strokes. We can see our method generates more vivid results with a clear distinction on brush textures, while the method "Learning-to-Paint" tends to produce blurred results. We also compare the stylized artworks created by our method with those created manually. Fig. 11 shows the comparison, where the second column shows the results create by a famous artist Adam Lister from New York. Those manual artworks are from his official gallery website. We can see both the manual result and automated results present a low-bit artistic geometry painting effect.
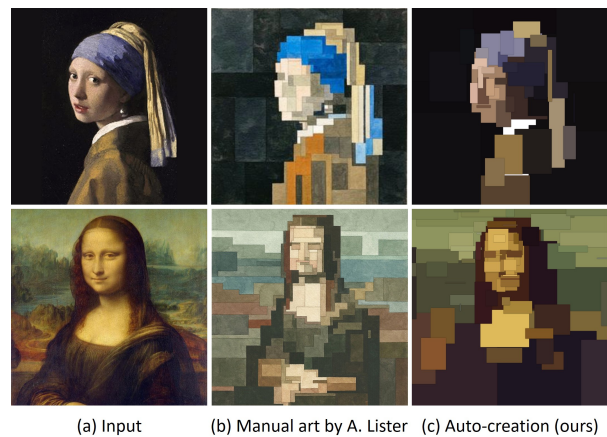
4.3. Controlled Experiments

Pixel loss vs transportation loss. To evaluate the effectiveness of our transportation loss function, we design the following experiment where we visually compare the painting generated by 1) pixel ℓ_1 loss only, and 2) pixel ℓ_1 loss + transportation loss. Fig. 12 shows the comparison result. We test on a flower picture and several digits from the MNIST dataset [16]. We can see that transportation loss helps to recover more details. Particularly, in the digit "1" and digit "4", using only the pixel ℓ_1 loss may produce bad convergence results on stroke parameters, which is similar to the failure case we have shown in Fig. 5.

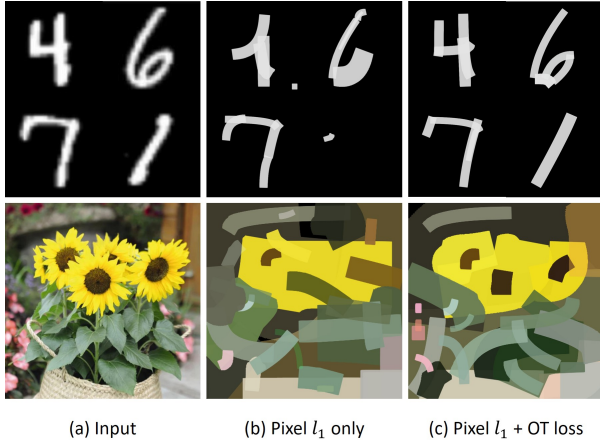**Compare with other neural renderers.** We compare our proposed renderer with some recent ones, including 1) a

(a) Input   (b) Pixel ℓ_1 only   (c) Pixel ℓ_1 + OT loss

Figure 12: A comparison between the strokes optimized by using pixel ℓ_1 loss only, and pixel ℓ_1 loss + optimal transportation loss on a flower picture and MNIST digits [16]. The transportation loss can help recover more details of the image, especially when the stroke is initialized with no overlap with the target region.

DCGAN-Generator-like renderer, which has been recently used for rendering 3D objects and faces [2, 29, 30, 37], and 2) a positional encoder + decoder architecture which has been used for rendering sketches and paintings [11, 35]. Here we refer to the former one as "DCGAN-G" and refer to the latter one as "PixShuffleNet". We compare their accuracy on randomly generated strokes. Fig. 13 shows the changes in their validation accuracy on different training epochs. We test on the watercolor brush and the oil-paint brush rendering tasks separately, but the marker-pen and tape-art renderer also have similar performances. All the renderers are trained under the same settings. We can see that in all brush settings, our neural renderer achieves the highest accuracy and is much better than other renderers ($+2.5\sim+5.0$ higher than others in PSNR). In addition, our renderer also has a much faster convergence speed than. In Fig. 14, we made a visual comparison between the different renderers. Our method produces the best result in terms of both color faithness and the high-frequency details. We also notice that the PxlShuffleNet fails to recover the color although we do have trained it on the color space. This could be caused by the highly coupled color and shape representations of its positional encoding or pixel-shuffle layers.

Rasterization network and shading network. In this experiment, we evaluate the effectiveness of the two components in our neural renderer: the rasterization network and the shading network. We design the following experiment where we first test on each of them separately and then test on our full implementation. All the networks are trained under the same settings. Table 1 shows the comparison results. We observe a noticeable accuracy drop (mean PSNR
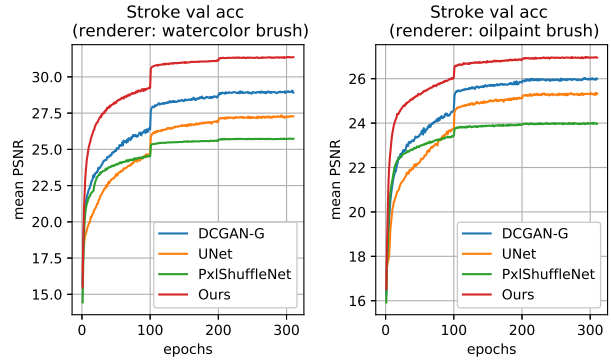

Figure 13: Validation accuracy (mean PSNR of the rendered foreground and alpha matte) of different neural renderers: DCGAN-G [29, 37], UNet [28], PxlShuffleNet [11]. Our render outperforms other renders with a large margin in both accuracy and convergence speed.
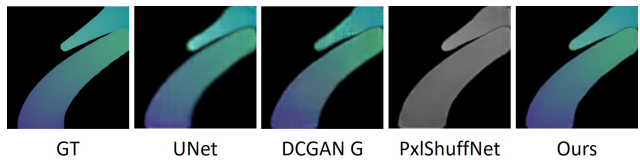

GT   UNet   DCGAN G   PxlShuffNet   Ours

Figure 14: A visual comparison between the results rendered by different neural renderers: UNet [28], DCGAN-G [2, 29], PxlShuffleNet [11], and ours.

Renderer/Stroke	Oil-paint	Watercolor
Ours (rasterization only)	24.015	25.769
Ours (shading only)	26.048	29.045
Ours (rasterization + shading)	26.982	31.389

Table 1: Ablation study of the proposed neural renderer.

of the foreground and alpha matte) when we remove either of the two networks from our renderer, which suggests the effectiveness of the dual-pathway design of our renderer.

5. Conclusion

We explore the nature of human painting by using differentiable stroke rendering. We consider this artistic creation process under a stroke parameter searching paradigm that maximizes the similarity between the sequentially rendered canvas and the reference image. Our method can generate highly realistic and painting artworks in vector format with controllable styles. We deal with the image similarity measurement from the perspective of optimal transportation and tackle the disentanglement of the color and shape with a dual-pathway neural renderer. Controlled experiments suggest the effectiveness of our design.

References

[1] Henry Abbott. Creating art using generative adversarial networks. 2019. 2

[2] Xuelin Chen, Daniel Cohen-Or, Baoquan Chen, and Niloy J Mitra. Neural graphics pipeline for controllable image generation. *arXiv preprint arXiv:2006.10569*, 2020. 8

[3] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013. 5

[4] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 3

[5] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018. 2, 3

[6] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016. 1, 2, 5

[7] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017. 2, 3

[8] Paul Haeberli. Paint by numbers: Abstract image representations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214, 1990. 2

[9] A Hertzmann. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, 2003. 3

[10] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8), 2012. 6

[11] Zhewei Huang, Wen Heng, and Shuchang Zhou. Learning to paint with model-based deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8709–8718, 2019. 2, 3, 5, 7, 8

[12] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 2

[13] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016. 1, 2

[14] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2018. 2

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 7, 8

[17] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photorealistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017. 2

[18] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. In *SIGGRAPH Asia 2018 Technical Papers*, page 222. ACM, 2018. 2

[19] Peter Litwinowicz. Processing images and video for an impressionist effect. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 407–414, 1997. 2

[20] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 2

[21] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer, 2014. 2

[22] Giulia Luise, Alessandro Rudi, Massimiliano Pontil, and Carlo Ciliberto. Differential properties of sinkhorn approximation for learning with wasserstein distance. In *Advances in Neural Information Processing Systems*, pages 5859–5870, 2018. 5

[23] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. 2015. 1, 2

[24] Reiichiro Nakano. Neural painters: A learned differentiable constraint for generating brushstroke paintings. *arXiv preprint arXiv:1904.08410*, 2019. 3

[25] Thu H Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. In *Advances in Neural Information Processing Systems*, pages 7891–7901, 2018. 3

[26] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019. 2

[27] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 5

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 5, 8

[29] Tianyang Shi, Yi Yuan, Changjie Fan, Zhengxia Zou, Zhenwei Shi, and Yong Liu. Face-to-parameter translation for game character auto-creation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 161–170, 2019. 2, 3, 8

[30] Tianyang Shi, Zhengxia Zou, Zhenwei Shi, and Yi Yuan. Neural rendering for game character auto-creation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*,

2020. 2, 8

[31] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016. 5

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5

[33] Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *IEICE TRANSACTIONS on Information and Systems*, 96(5):1134–1144, 2013. 2, 3

[34] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 4

[35] Ningyuan Zheng, Yifan Jiang, and Dingjiang Huang. Strokenet: A neural painting environment. In *International Conference on Learning Representations*, 2018. 3, 8

[36] Tao Zhou, Chen Fang, Zhaowen Wang, Jimei Yang, Byungmoon Kim, Zhili Chen, Jonathan Brandt, and Demetri Terzopoulos. Learning to sketch with deep q networks and demonstrated strokes. *arXiv preprint arXiv:1810.05977*, 2018. 2, 3

[37] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European conference on computer vision*, pages 597–613. Springer, 2016. 3, 8

[38] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017. 1, 2