УДК 519.85:004.42

M. Poliakov, N. Shvai

DOI: 10.18523/2617-70807202435-43

GAN-GENERATED STROKES EXTENSION FOR PAINT TRANSFORMER

Neural painting produces a sequence of strokes for a given image and artistically recreates it using neural networks. In this paper, we explore a novel Transformer-based framework named the Paint Transformer to predict the parameters of a stroke set with a feed-forward neural network. The Paint Transformer achieves better painting results than previous methods with more inexpensive training and inference costs. The paper proposes a novel extension to the Paint Transformer that adds more complex GAN-generated strokes to achieve a more artistically abstract painting style than the original method. This research was originally published as a Master's thesis [1].

Keywords: neural painting, transformer, GAN.

Introduction

Painting has been an excellent way for humans to record what they perceive or even imagine the universe around them and has long been known to demand proficiency. Computer-aided art design essentially fills this gap and enables us to make our creative pieces, particularly with the appearance of AI. However, most current generative AI painting methods are still centered on teaching computers how to "paint" at the pixel level to achieve or mimic some painting style, for example, purely GAN-based approaches [2] and style transfer [3]. Humans create artworks through a stroke-by-stroke process, using brushes from coarse to fine. It is of great potential to make machines imitate such a stroke-by-stroke process to develop more genuine and human-like paintings. Thus, as an emerging research topic, stroke-based neural painting is analyzed to generate a series of strokes to mimic how human painters create artistic works. Generating stroke sequences for the painting process is challenging even for skilled human painters, especially when the targets have complicated compositions and rich textures. Some previous work tackles this problem by a sequential process of generating strokes one by one, such as greedy search step-by-step [4], recurrent neural networks [5], and reinforcement learning [6]. Using an iterative optimization process, techniques [7] tackle this problem via stroke parameter searching.

Although these methods generate attractive painting results, considerable room for advancement in both efficiency and effectiveness still exists. Sequence-based methods such as RL are relatively quick in inference but suffer from lengthy training time and unstable agents. Meanwhile, optimization-based approaches do not need training, but their optimization process is highly time-consuming. Distinct from earlier techniques, in this paper, we explore the painting process as a set prediction task and a novel Transformer-based framework, named Paint Transformer, proposed by [8], to predict the parameters of a stroke set with a feed-forward neural network. Paint Transformer achieves better painting results than previous methods with more inexpensive training and inference costs.

Despite excellent Paint Transformer results, there is room for further improvement. The paper proposes a novel extension to Paint Transformer that adds more complex GAN-generated strokes to achieve a more artistically abstract painting style than the original method.

Related work

Paint Transformer. Paint Transformer is a progressive stroke prediction procedure. The model predicts multiple strokes in parallel at each step to minimize the difference between the current canvas and our target image. Paint Transformer has two modules: *Stroke Renderer* and *Stroke Predictor*. Provided a target picture, I_t, and an intermediate canvas picture, I_c, *Stroke Predictor* yields a set of parameters to choose the current stroke set S_r. Then, *Stroke Renderer* renders the stroke picture for each stroke in S_r and plots them onto the canvas I_c, creating the resultant image I_r [8]. Or simply:

$$I_r = PaintTransformer(I_c, I_t) \qquad (1)$$

Only *Stroke Predictor* is trainable in Paint Transformer, while *Stroke Renderer* is a parameter-free and differentiable module. *Stroke Predictor* has a self-training pipeline that uses randomly sampled strokes. During training, in each iteration, a

list of foreground stroke parameters S_f and a list of background stroke parameters S_b are randomly sampled. *Stroke Renderer* then generates a canvas picture I_c by taking as input S_b and producing a target picture I_t by overlaying S_f onto I_c. In the end, *Stroke Predictor* taking I_c and I_t as input can predict a stroke list S_r, after which *Stroke Renderer* can produce a predicted image I_r taking S_r and I_c as intake. Therefore, *Stroke Predictor* optimization is conducted on both stroke and pixel levels and the training goal can be stated as:

$$\mathcal{L} = \mathcal{L}_{stroke}(S_r, S_f) + \mathcal{L}_{pixel}(I_r, I_t) \qquad (2)$$

where $\mathcal{L}_{pixel}$ and $\mathcal{L}_{stroke}$ are pixel loss and stroke loss, respectively. Strokes are randomly sampled so that unlimited data for training can be generated. Thus, Paint Transformer does not need any prepared-in-advance training dataset.

In Paint Transformer, a stroke is a simple 1-channel brush image, called primitive brush, which can be transformed by shape parameters and color parameters. The shape parameters of a stroke include height h, width w; a center point coordinates x, y, and rotation angle θ. Color parameters contain RGB values represented as r, g, and b. Thus, a stroke s can be denoted as $\{x, y, h, w, \theta, r, g, b\}$. Let I_{in} and I_{out} be an input and output canvases, and $S = \{s_i\}_{i=1}^n$ is a list of n strokes. Given a primitive brush image I_b and a stroke s_i, *Stroke Renderer* can change its color, and affine transforms its shape and location in the canvas Cartesian coordinate system, obtaining its rendered stroke image $\bar{I}_b^i$. Also, the renderer generates a 1-channel alpha map α_i with the same shape of $\bar{I}_b^i$, as a binary mask of s_i. Representing $I_{mid}^0 = I_{in}$ and $I_{mid}^n = I_{out}$, it is possible to write the stroke rendering operation:

$$I_{mid}^i = \alpha^i \cdot \bar{I}_b^i + \left(1 - \alpha^i\right) \cdot I_{mid}^{i-1} \qquad (3)$$

and the whole *Stroke Renderer* process as:

$$I_{out} = StrokeRenderer(I_{in}, S) \qquad (4)$$

The purpose of a *Stroke Predictor* is to predict a set of strokes that can cover the distinctions between an intermediate target and a canvas image. Taking I_c, I_t with dimension 3 x m x m as input (here, m is the stroke image's width and height, and 3 is the number of channels), *Stroke Predictor* passes I_c and I_t through two independent convolution neural networks to extract their feature maps as F_c, F_t with dimension c $\times$ $\frac{m}{4}$ $\times$ $\frac{m}{4}$.

Afterward, F_c, F_t, and a learnable positional encoding [9] are concatenated and flattened as the intake of the Transformer encoder. The decoder part takes N learnable stroke vectors as intake. Ultimately, the decoder predicts initial stroke parameters $\bar{S}_r = \{s_i\}_{i=1}^N$ and stroke confidence $C_r = \{c_i\}_{i=1}^N$ using two branches of fully-connected layers.

Furthermore, in the forward phase, confidence score c_i can be transformed to a decision $d_i = BinarySign(c_i)$, where *BinarySign* is a binary function whose value is 1 if c_i is positive and is 0 otherwise. The decision d_i is used to decide whether a predicted stroke should be painted on the canvas image. Because the *BinarySign* function has zero gradient almost everywhere to enable backpropagation sigmoid function $\sigma(x)$ is used to compute the gradient [8]:

$$\frac{\partial d_i}{\partial c_i} = \frac{\partial \sigma\left(c_i\right)}{\partial c_i} = \frac{\exp\left(-c_i\right)}{\left(1 + \exp\left(-c_i\right)\right)^2} \qquad (5)$$

Collecting all inferred strokes with positive decisions, it is possible to get the final $S_r = \{s_i\}_{i=1}^N$ with N strokes and define *Stroke Predictor* as:

$$S_r = StrokePredictor(I_c, I_t) \qquad (6)$$

Paint Transformer can simultaneously minimize the differences between target and prediction on both image and stroke levels. Here is a list of losses that are used to train Paint Transformer [8]: Pixel loss is straightforwardly used to recreate a target image. Thus, the pixel-wise loss $\mathcal{L}_{pixel}$ between I_r and I_t is minimized on the whole image level:

$$\mathcal{L}_{pixel} = \left\|I_r - I_t\right\|_1 \qquad (7)$$

Stroke loss is essential to define suitable metric for estimating the difference between strokes on the stroke level. To achieve the best results, three-component losses are combined in the loss. Stroke $\mathcal{L}_\infty$ distance is intuitive (s_u and s_v indicate parameters of strokes u and v, respectively):

$$\mathcal{D}_{L_1}^{u,v} = \left\|s_u - s_v\right\|_1 \qquad (8)$$

Wasserstein distance is used because using the L_1 metric alone ignores different scales for large and little strokes. A rectangular rotational stroke with parameters $\{x, y, h, w, \theta\}$ can be represented as a 2D Gaussian distribution $N(\mu, \sum)$ by the next equations as stated in [10]:

$$\begin{aligned} &\mu = (x, y), \\ &\Sigma^{\frac{1}{2}} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \frac{w}{2} & 0 \\ 0 & \frac{h}{2} \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} \frac{w}{2}\cos^2\theta + \frac{h}{2}\sin^2\theta & \frac{w-h}{2}\cos\theta\sin\theta \\ \frac{w-h}{2}\cos\theta\sin\theta & \frac{w}{2}\sin^2\theta + \frac{h}{2}\cos^2\theta \end{bmatrix} \end{aligned} \qquad (9)$$

Hence, the Wasserstein distance between Gaussian distributions $N(\mu_u, \sum_u)$ and $N(\mu_v, \sum_v)$ is (where Tr denotes the trace of a matrix):

$$\mathcal{D}_W^{u,v} = \left\|\mu_u - \mu_v\right\|_2^2 + \mathrm{Tr}\left(\Sigma_u + \Sigma_v - 2\left(\Sigma_u^{\frac{1}{2}}\Sigma_v\Sigma_u^{\frac{1}{2}}\right)^{\frac{1}{2}}\right) \qquad (10)$$

Binary cross-entropy is used to predict a stroke's confidence with the positive (negative) ground-truth decision should be as high (low) as possible. Let's assume s_v as a target stroke with ground-truth label g_v and s_u as a predicted stroke with confidence c_u and, where $g_v = 0$ if s_v is an empty stroke and $g_v = 1$ if s_v is a valid stroke:

$$\mathcal{D}_{bce}^{u,v} = -g_v \cdot \log \sigma(c_u) - (1 - g_v) \cdot \log(1 - \sigma(c_u)) \quad (11)$$

The number of valid ground-truth strokes varies during training. Paint Transformer has a matching instrument between the prediction set $\bar{S}_r$ of N strokes and the ground-truth set S_g of a maximum N strokes (there could be both empty and valid strokes in S_g) to compute the loss function. Paint Transformer uses the permutation of strokes that yields the minimal stroke-level matching cost to calculate final loss using the Hungarian algorithm. For prediction set $\bar{S}_r$ that has a stroke s_u and for the target set S_g that has a stroke s_v, their cost value is (corresponding cost for empty target strokes is always 0):

$$M_{u,v} = g_v(\mathcal{D}_{L_1}^{u,v} + \mathcal{D}_{W}^{u,v} + \mathcal{D}_{bce}^{u,v}) \quad (12)$$

Thus, marking the optimal permutations for predicted and target strokes as X and Y, respectively, that are provided by the Hungarian algorithm, respectively, the stroke loss is given by:

$$\mathcal{L}_{stroke} = \frac{1}{n} \sum_{i=1}^{n} g_{Y_i}(\mathcal{D}_{L_1}^{X_i Y_i} + \mathcal{D}_{W}^{X_i Y_i} + \mathcal{D}_{bce}^{X_i Y_i}) \quad (13)$$

Paint Transformer imitates a coarse-to-fine algorithm to mimic an artist and yield painting results during prediction. Provided a photo of dimension $H \times W$, Paint Transformer runs from coarse to fine in order on K rankings. Painting on each ranking is conditional on the result of the prior ranking. The target image and current canvas are cut into the number of non-overlapping $P \times P$ patches before being processed by the *Stroke Predictor*.

Neural Painters. The paper by Reiichiro Nakano investigates different experiments with neural painters built on differentiable simulations of a non-differentiable painting program. Firstly, two methods of training a neural painter using VAEs and GANs, respectively, are presented. Secondly, the paper recreates *SPIRAL* reconstruction results [11] using a non-RL learning adversarial technique with a neural painter. Thirdly, the use of a neural painter as a differentiable image parameterization is suggested. By optimizing strokes directly using backpropagation, a method is suggested to visualize pre-trained image classifiers by letting them to paint classes they were trained to determine [12]. For the purposes of this paper, we are specifically interested in the GAN-reconstruction of a non-differentiable painting program brushstrokes.

The action space represents the set of parameters that are used as control inputs for the MyPaint. The action space maps a single action to a single stroke in the MyPaint. An agent paints by sequentially yielding actions and spreading full strokes on a canvas. The action space consists of the next parameters [12]:

- Brush coordinates are a set of three Cartesian coordinates pairs representing the stroke shape. The coordinates describe a start point, end point, and middle control point, forming a quadratic Bezier curve. We denote them as $\{x_s, y_s, x_e, y_e, x_c, y_c\}$ respectively.
- Start and end pressure describe the pressure used on the brush at the start and end of the stroke. We denote them as $\{p_s, p_e\}$ respectively.
- Brush size that specifies the brush radius and denoted as s.
- Color consists of three variables that represent the RGB color of the brush and specified as $\{r, g, b\}$.

To recreate a MyPaint brushstroke using a neural network [12] proposes VAE and GAN methods. We focus here on the GAN [13] method because it produces sharper images than VAE and thus more accurate strokes. An adversarial loss is used to directly learn a mapping from actions to strokes. Unlike a typical GAN, the noise is not injected into the intake of the generator. Instead, the generator takes the input action and maps it directly to a stroke. The discriminator is provided with real and generated action-stroke pairs and tries to decide whether the pair is real. This is comparable to a conditional GAN [14]. Pairs of true strokes on the left and the complementary GAN neural painter results on the right.

Methodology

GAN-generated strokes training. In this paper, we propose combining GAN-stroke rendering system referenced above with Paint Transformer to introduce more complex strokes. Current Paint Transformer *Stroke Renderer* has only eight parameters, while GAN-stroke rendering has 12 with potential to increase up to 50 parameters that MyPaint supports. First of all, we needed to set up the MyPaint program to generate strokes for the training, so we prepared a setup script for macOS [15]. The training of the GAN network was done locally on an Intel CPU based Mac laptop. Since MyPaint generated the strokes using CPU,

there is a bottleneck for training performance even if GPU is available. The training code is based on the implementation of Neural Painters by [12]. The modification is that we directly feed generated MyPaint strokes into the discriminator in real-time from a data loader. At the same time, [12] pre-generated stroke images first and trained the network afterward. We also simplified the MyPaint API calling code and wrapped it in a data loader [15] for convenience. The following stroke parameters are used: $\{x_s, y_s, x_e, y_e, x_c, y_c, s, p_s, p_e\}$. We do not use color parameters to train strokes compared to the original implementation because we can colorize the strokes later in the Paint Transformer. We trained GAN strokes for about 11.7 million iterations, and it took about 36 hours to do so due to the CPU bottleneck.

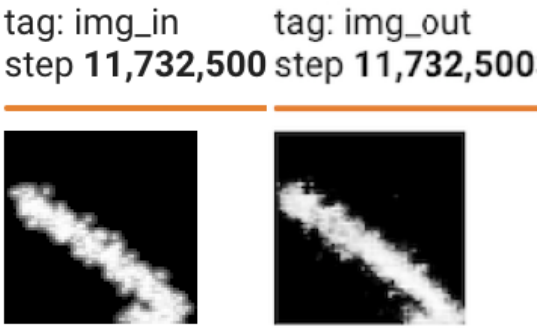


Figure 1. A GAN result sample on 11.7 million iterations

The sample of the result on the final iteration is provided in Figure 1. On the left (img_in) is the image painted by MyPaint and on the right (img_out) is the GAN-generated image on the same set of action parameters. The discriminator loss, generator score, and real score are provided in Figures 2, 3, and 4, respectively. The x-axis depicts iterations, and the y-axis is the numeric score.

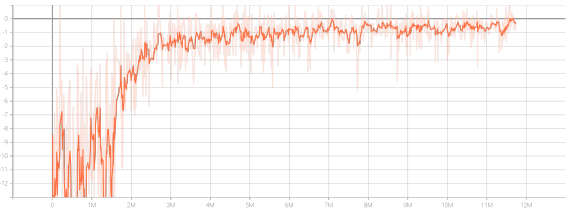


Figure 2. GAN-generated strokes discriminator loss

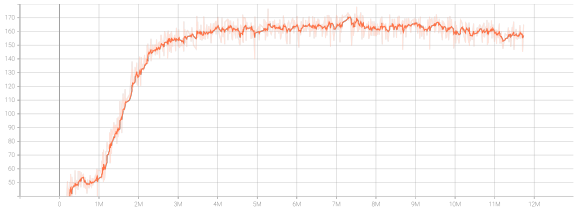


Figure 3. GAN-generated strokes generator score

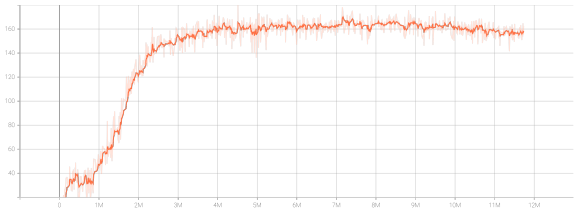


Figure 4. GAN-generated strokes real score

GAN Stroke Renderer. Once the GAN stroke predictor is trained, we can quickly predict the MyPaint stroke shape using nine parameters on the GPU with comparable quality to MyPaint. We can now utilize the GAN-generated strokes we trained, as a basis for a new stroke renderer for Paint Transformer that would yield more advanced strokes than the original. We denote the new stroke renderer as a *GAN Stroke Renderer*. The set of parameters is now $\{x_s, y_s, x_e, y_e, x_c, y_c, s, p_s, p_e, r, g, b\}$. As a first step, we infer set stroke shapes from $\{x_s, y_s, x_e, y_e, x_c, y_c, s, p_s, p_e\}$ using GAN-generated strokes pre-trained weights (step 1 denoted in Figure 5). Note that we do not train GAN-generated strokes anymore and use them as a predictor. The GAN output does not have clear zero pixels and has numbers close to zero instead. Therefore, we cannot create a binary mask immediately and must utilize a denoising solution. Considering Equation 3 for original *Stroke Renderer*, we form an alpha map via $\alpha^i = \bar{I}_b^i > > Q_{0.8}(\bar{I}_b^i)$, where $Q_{0.8}$ is 80th-percentile. By also forming a color map c^i from $\{r, g, b\}$ we can rewrite Equation 3 as (steps 2, 3 denoted in Figure 5):

$$I_{mid}^i = \alpha^i \cdot \bar{I}_b^i \cdot c^i + \left(1 - \alpha^i\right) \cdot I_{mid}^{i-1} \qquad (14)$$

Stroke Predictor modification. We would need to modify the Stroke Predictor so that it could work with the new GAN Stroke Renderer. We change the architecture of the Transformer to accept 12 parameters instead of 8 original. The main challenge is to modify the loss function so that it would take new parameters. Specifically, the Wasserstein distance needs a modification (Equation 9). Initially,

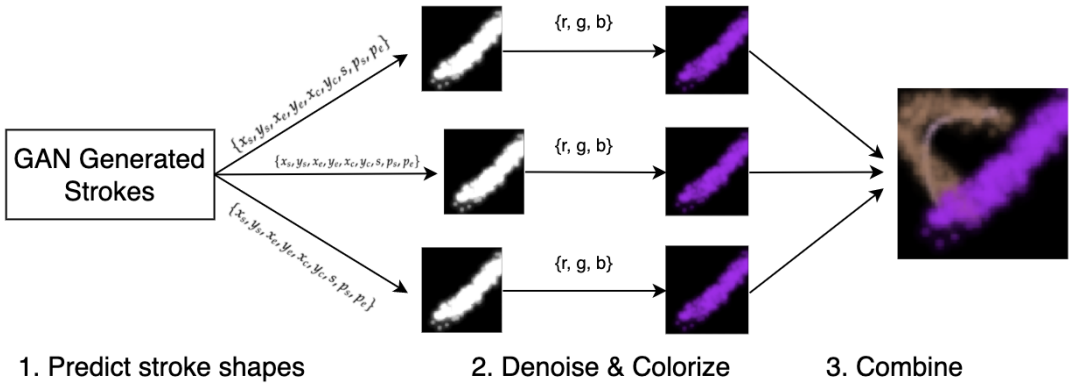
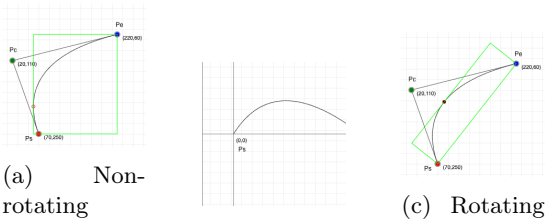


Figure 5. *GAN Stroke Renderer*

it accepts $\{x, y, h, w, \theta\}$, and we need to take $\{x_s, y_s, x_e, y_e, x_c, y_c, s, p_s, p_e\}$. Instead of modifying this loss function directly, we decided instead to translate $\{x_s, y_s, x_e, y_e, x_c, y_c, s, p_s, p_e\}$ into $\{x, y, h, w, \theta\}$ by creating a rotating bounding box around the stroke.

We know that the stroke is a quadratic Bezier curve represented by, where $P_0 = P_s = (x_s, y_s)$, $P_1 = P_c = (x_c, y_c)$, $P_2 = P_e = (x_e, y_e)$ and $t \in \in [0, 1]$:

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{2} B_i^2(t) \cdot P_i = \sum_{i=0}^{2} t^i (1-t)^{2-i} \cdot P_i \\ &= (1-t)^2 \cdot P_0 + 2t(1-t) \cdot P_1 + t^2 \cdot P_2 \\ &= (1-t)^2 \cdot P_s + 2t(1-t) \cdot P_c + t^2 \cdot P_e\end{aligned} \tag{15}$$



(a) Non-rotating bounding box (b) Curve alignment (c) Rotating bounding box

Figure 6. Bezier curve [16]

Firstly, we can find a non-rotating bounding box by finding extremities of the Bezier curve by finding maxima and minima on the component functions, solving the equation $\mathbf{B}'(t) = 0$ [16]:

$$\mathbf{B}'(t) = 2(1-t)(P_c - P_s) + 2t(P_e - P_c) = 0 \implies t = \frac{P_s - P_c}{-2 \cdot P_c + P_s + P_e} \tag{16}$$

Now when we know t, we could find the solution and compare it with P_s and P_e. The lowest value is the lower point $P_{min} = min(\mathbf{B}(t), P_s, P_e)$, and the highest is the upper point for the bounding box $P_{max} = max(\mathbf{B}(t), P_s, P_e)$ (Figure 5a). To get a rotated bounding box, we need to make $P_s = (0, 0)$ and align the curve on the x-axis via (Figure 5b):

$$\begin{aligned}&\alpha = \arctan\frac{y_e}{x_e} \implies R = \begin{bmatrix} \cos(-\alpha) & -\sin(-\alpha) \\ \sin(-\alpha) & \cos(-\alpha) \end{bmatrix} \\ &\dot{P}_s = P_s - P_s = (0, 0) \\ &\dot{P}_c = (P_c - P_s) \cdot R \\ &\dot{P}_e = (P_e - P_s) \cdot R\end{aligned} \tag{17}$$

Afterward, we calculate a non-rotating bounding box for $\dot{P}_s, \dot{P}_c, \dot{P}_e$ via Equations 15, 16 and make a reverse transformation [16]:

$$\begin{aligned}\dot{P}_{min} &= min(\dot{\mathbf{B}}(t), \dot{P}_s, \dot{P}_e) = (\dot{x}_{min}, \dot{y}_{min}) \\ \dot{P}_{max} &= max(\dot{\mathbf{B}}(t), \dot{P}_s, \dot{P}_e) = (\dot{x}_{max}, \dot{y}_{max}) \\ (x_{max}, y_{max}) &= (\dot{x}_{max}, \dot{y}_{max}) \cdot R^{-1} + P_s \\ (x_{min}, y_{min}) &= (\dot{x}_{min}, \dot{y}_{min}) \cdot R^{-1} + P_s \\ (x'_{max}, y'_{max}) &= (\dot{x}_{max}, \dot{y}_{min}) \cdot R^{-1} + P_s \\ (x'_{min}, y'_{min}) &= (\dot{x}_{min}, \dot{y}_{max}) \cdot R^{-1} + P_s\end{aligned} \tag{18}$$

Thus, a rotating bounding box can be represented by four points $(x_{min}, y_{min}), (x_{max}, y_{max}), (x'_{min}, y'_{min}), (x'_{max}, y'_{max})$ as depicted in Figure 5c. We also need to account for start and end pressure and size; we found empirically that we can modify $\{x_s, y_s, x_e, y_e, x_c, y_c\}$ with $\{s, p_s, p_e\}$ before calculating the bounding box, which yields better results (clamp is used to

restrict a value between 0 and 1):

$$\begin{aligned} x_s &= \text{clamp}(x_s + 0.15 \cdot p_s,\ 0,\ 1); \\ y_s &= \text{clamp}(y_s + 0.15 \cdot p_s,\ 0,\ 1) \\ x_e &= \text{clamp}(x_e + 0.15 \cdot p_e,\ 0,\ 1); \\ y_e &= \text{clamp}(y_s + 0.15 \cdot p_e,\ 0,\ 1) \\ x_c &= \text{clamp}(x_c - 0.15 \cdot s,\ 0,\ 1); \\ y_c &= \text{clamp}(y_c - 0.15 \cdot s,\ 0,\ 1) \end{aligned} \tag{19}$$

Finally, we need to convert four coordinate points into $\{x, y, h, w, \theta\}$:

$$\begin{aligned} x &= \frac{x_{max} + x_{min}}{2}; \quad y = \frac{y'_{max} + y_{min}}{2} \\ w &= \sqrt{(y'_{max} - y_{max})^2 + (x'_{min} - x_{max})^2} \\ h &= \sqrt{(y_{max} - y'_{min})^2 + (x_{max} - x'_{max})^2} \\ \theta &= \arctan \frac{y'_{max} - y_{max}}{x'_{max} - x_{max}} \end{aligned} \tag{20}$$

The code implementation can be found in `get_rotated_bounding_box` [15] method, and the resulting bounding boxes on GAN-generated strokes are shown in Figure 7.

Figure 7. Rotated bounding box on GAN-generated strokes

Experiments

Training. Once the *Stroke Predictor* is optimized for the new set of parameters, we can train Paint Transformer with a *GAN Stroke Renderer* system. We trained Paint Transformer for 180 epochs, and the training process took about 6 hours on NVIDIA RTX 5000. In comparison, the original Paint Transformer takes about 5-5.5 hours to train 180 epochs on the same GPU. This means that the training time for our GAN extension did not increase significantly. In Figures 8 and 9, we can notice the training charts for pixel and stroke L_1 distance losses. Wasserstein distance loss and a binary cross-entropy decision loss are depicted in Figures 10 and 11, respectively. The x-axis depicts iterations, and the y-axis shows the numeric score. Canvas-target-predict triads S_b, S_f, and S_r, are shown in Figure 12 during the training. We also changed the monitoring framework from Visdom to the commonly used Tensorboard.

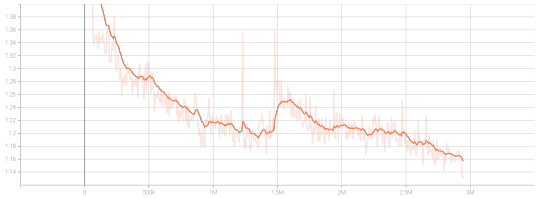


Figure 8. Paint Transformer pixel loss

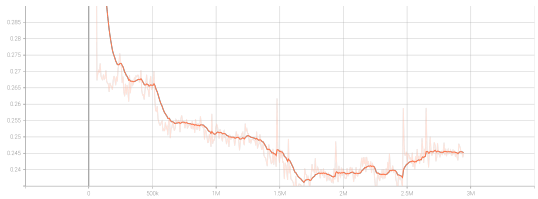


Figure 9. Paint Transformer stroke L_1 distance loss

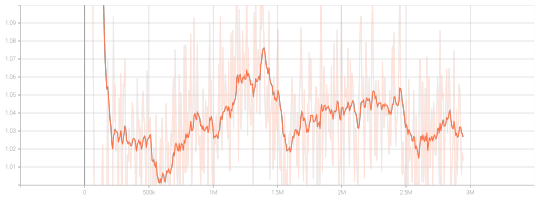


Figure 10. Paint Transformer Wasserstein distance loss

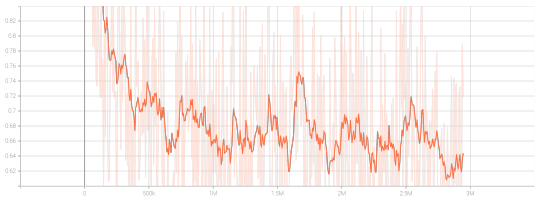


Figure 11. Paint Transformer binary cross-entropy decision loss

Results. We modified the inference module so it could work with *GAN Stroke Renderer* and obtained the results depicted in Figure 13c. We take Figure 13a as an input, and we also show the original Paint Transformer results in Figure 13b. For comparison, we choose the images in different settings: sunflower and frog are macro images, while the fjord and the city are landscape images. Overall, we can notice that our Paint Transformer extension paints the resulting pictures in a more granular fashion (using the same value of K as the original). This creates a more abstract painting style, especially in the fjord, city, and sunflower cases. We can also notice that the Paint Transformer extension does not paint larger strokes for uniform patches. We can also notice that the Paint Transformer extension does not paint larger

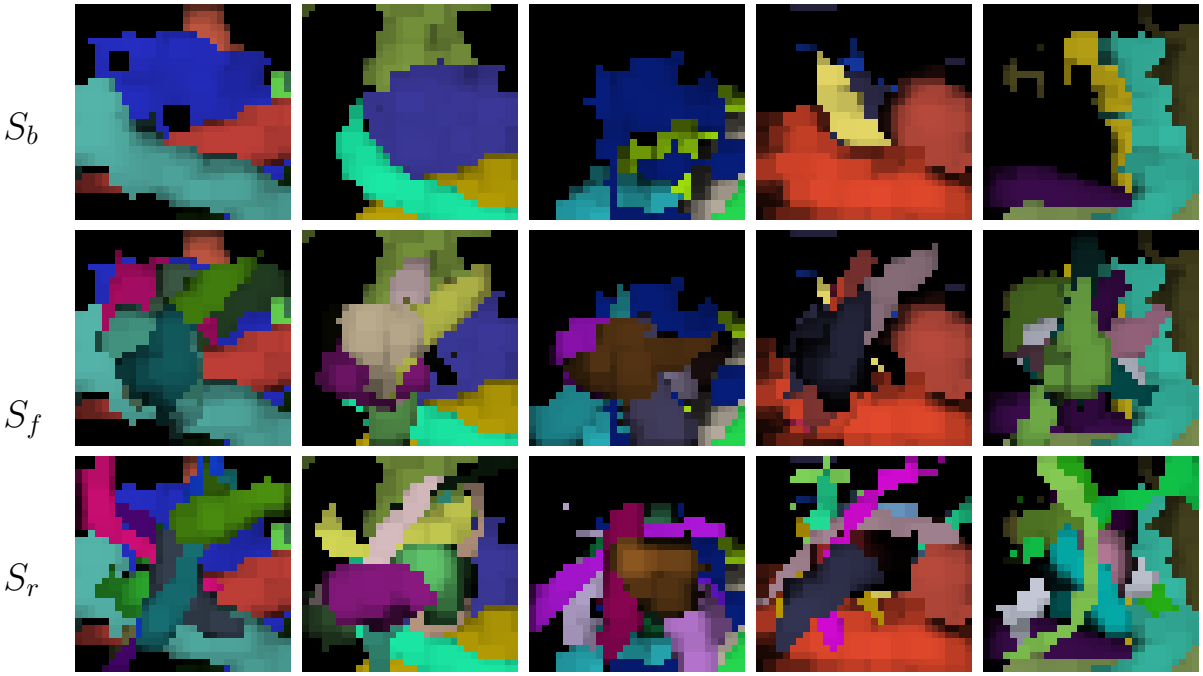
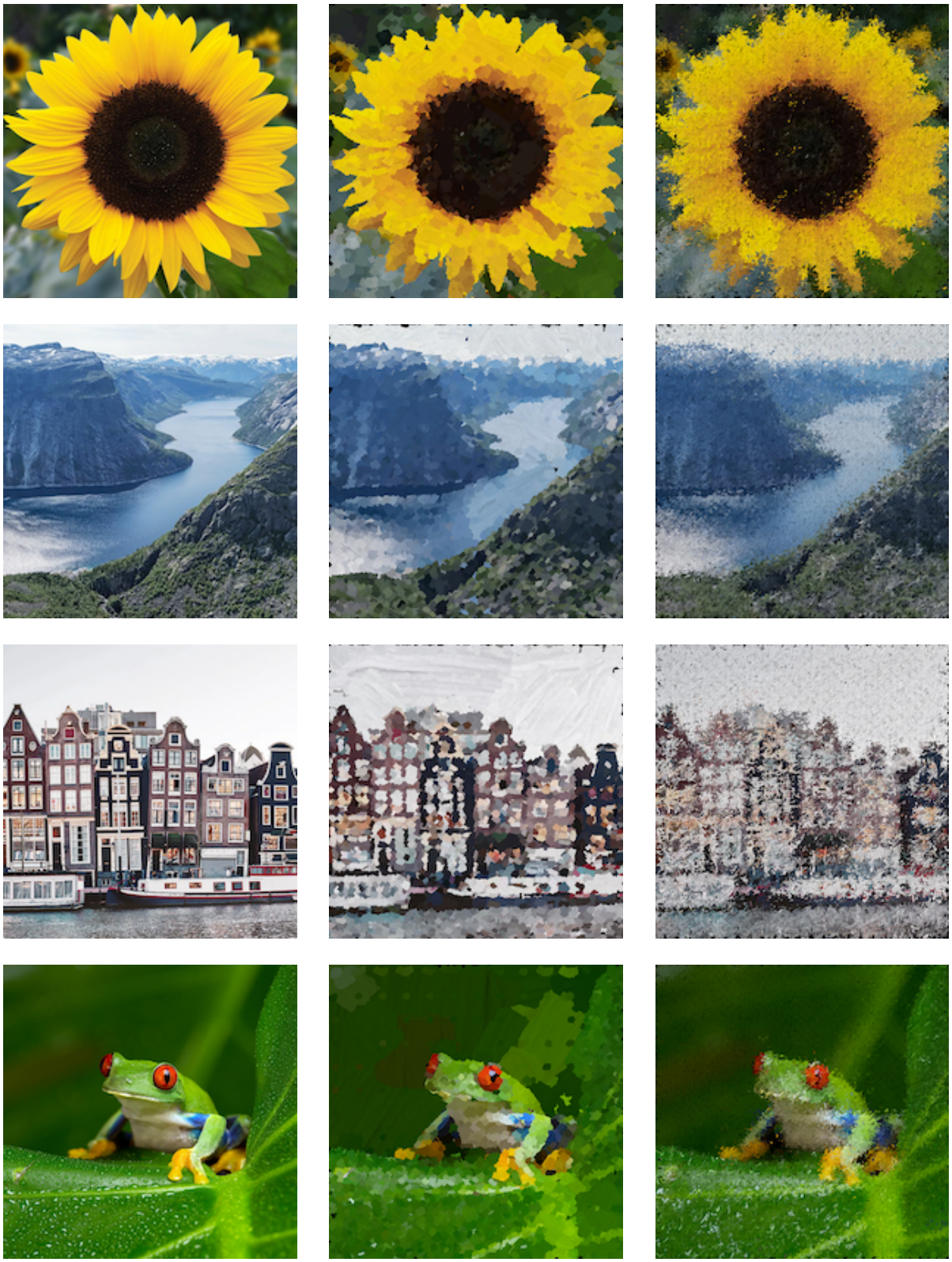


Figure 12. Canvas-target-predict triads in training

strokes for the uniform patches. We need to further modify Wasserstein distance loss and the binary cross-entropy decision loss to improve the results.

Conclusions

We proposed a GAN strokes extension to the Paint Transformer aimed at introducing more complex strokes. We refined the Stroke Rendering system, which generates strokes using a pre-trained GAN and has 12 parameters compared to the original 8. We partly modified the loss function to accept a new parameter list. The results have a different painting style and are more abstract; however, the extension paints strokes of similar size. This indicates that we need to make further effort in modifying Wasserstein distance loss and the binary cross-entropy decision loss to improve the results, which we plan to address in our future work. In addition, converting the network architecture to use 4-channel images might further enhance the results by removing artifacts on the generated strokes.



(a) Input image (b) Original output (c) GAN-extension output

Figure 13. Comparison of the input image, original output and GAN-extension output.

References

1. M. Poliakov, https://ekmair.ukma.edu.ua/handle/123456789/28820.
2. A. Elgammal, https://arxiv.org/abs/1706.07068.
3. L. A. Gatys, https://arxiv.org/abs/1508.06576.
4. P. Haeberli, in: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques. — SIGGRAPH '90 (New York, NY, USA: Association for Computing Machinery, 1990), pp. 207–214. https://doi.org/10.1145/97879.97902.
5. D. Ha, https://arxiv.org/abs/1704.03477.
6. T. Zhou, https://arxiv.org/abs/1810.05977.
7. Z. Zou, https://arxiv.org/abs/2011.08114.
8. S. Liu, https://arxiv.org/abs/2108.03798.
9. A. Vaswani, https://arxiv.org/abs/1706.03762.
10. X. Yang, https://arxiv.org/abs/2101.11952.
11. Y. Ganin, https://arxiv.org/abs/1804.01118.
12. R. Nakano, https://arxiv.org/abs/1904.08410.
13. I. J. Goodfellow, https://arxiv.org/abs/1406.2661.
14. M. Mirza, https://arxiv.org/abs/1411.1784.
15. M. Poliakov, https://github.com/mxpoliakov/PaintTransformerGAN.
16. Pomax. A primer on bezier curves, https://pomax.github.io/bezierinfo.

Поляков М. Х., Швай Н. О.

РОЗШИРЕННЯ МОЖЛИВОСТЕЙ PAINT TRANSFORMER З ГЕНЕРУВАННЯМ МАЗКІВ ПЕНЗЛЯ ЗА ДОПОМОГОЮ GAN

Нейронне малювання створює послідовність мазків для заданого зображення і художньо відтворює його за допомогою нейронних мереж. У цій статті ми досліджуємо нову архітектуру, основану на Transformer, під назвою Paint Transformer, яка прогнозує параметри набору мазків за допомогою прямопрохідної нейронної мережі. Paint Transformer забезпечує кращі результати малювання порівняно з попередніми методами, маючи нижчі витрати на навчання та використання. У статті також пропонується нове розширення Paint Transformer, яке додає більш складні мазки, згенеровані GAN, для досягнення більш художнього та абстрактного стилю малювання, ніж оригінальний метод.

Ключові слова: нейронне малювання, трансформер, GAN.

Матеріал надійшов 07.01.2025