Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему: **«Розширення можливостей Paint Transformer з генеруванням мазків пензля за допомогою GAN»**

Виконав: студент 2-го року навчання
освітньо-наукової програми
«Прикладна математика»,
спеціальності 113 Прикладна
математика

Поляков Михайло Хельгович

Керівник: Швай Н. О.,
кандидат фіз.-мат. наук, доцент

Рецензент_____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 20____ р.

Київ – 2022

GAN-generated strokes extension for Paint Transformer

by Mykhailo Poliakov

Abstract

Neural painting produces a sequence of strokes for a given image and artistically recreates it using neural networks. In this thesis, we explore a novel Transformer-based framework named Paint Transformer 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. The thesis 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.

Contents

List of Figures

List of Abbreviations

RL	**R**einforcement **L**earning
GAN	**G**enerative **A**dversarial **N**etwork
VAE	**V**ariational **A**uto**e**ncoder
RNN	**R**ecurrent **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphics **P**rocessing **U**nit
RAM	**R**andom **A**ccess **M**emory
RGB	**R**ed **G**reen **B**lue
ReLU	**Re**ctified **L**inear **U**nit
API	**A**pplication **P**rogramming **I**nterface

1 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. Nevertheless, most of the 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 GANs-based approaches (Elgammal et al., 2017) and style transfer (Gatys et al., 2015). 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 imitate 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 works tackle this problem by a sequential process of generating strokes one by one, such as step-wise greedy search (Haeberli, 1990), recurrent neural networks (Ha & Eck, 2017), and reinforcement learning (Zhou et al., 2018). Using an iterative optimization process, techniques (Zou et al., 2020) 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 thesis, we explore the painting process as a set prediction task and a novel Transformer-based framework, named Paint Transformer, proposed by Liu et al., 2021, 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 thesis 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. We explore Paint Transformer as a main related work in Section 2.1 and a paper named "Neural Painters: A learned differentiable constraint for generating brushstrokes" by Nakano, 2019 in Section 2.2 as a supporting related work. This paper provides the basis of GAN-generated brushstrokes. In Chapter 3, we train GAN-generated strokes and combine them with Paint Transformer architecture to improve its results. Chapter 4 concludes the thesis.

2 Related works

2.1 Paint Transformer

2.1.1 Overall Framework

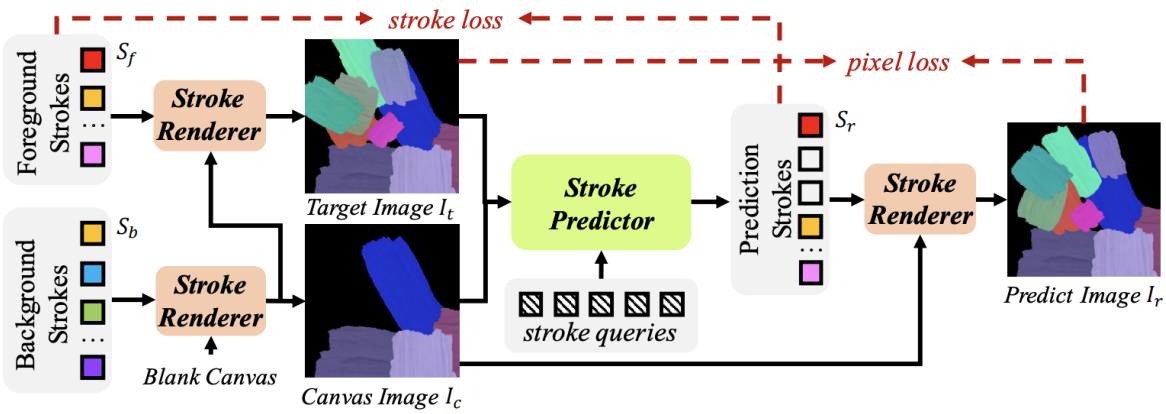


FIGURE 2.1: Self-training pipeline schema (Liu et al., 2021, p. 3)

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 params to choose the current stroke set S_r. Then, *Stroke Renderer* renders the stroke picture for each stroke in Sr and plots them onto the canvas I_c, creating the resultant image I_r (Liu et al., 2021, p. 3). Or simply:

$$I_r = PaintTransformer(I_c, I_t) \tag{2.1}$$

Only *Stroke Predictor* is trainable in Paint Transformer, while *Stroke Renderer* is a parameter-free and differentiable module. As demonstrated in Figure 2.1, *Stroke Predictor* has a self-training pipeline that uses randomly sampled strokes. During training, in each iteration, a foreground stroke parameter list S_f and a background stroke parameter list S_b is 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. So, *Stroke Predictor* optimization is conducted on both stroke and pixel levels and the training goal can be stated as:

$$\mathscr{L} = \mathscr{L}_{stroke}(S_r, S_f) + \mathscr{L}_{pixel}(I_r, I_t) \tag{2.2}$$

where $\mathscr{L}_{pixel}$ and $\mathscr{L}_{stroke}$ are pixel loss and stroke loss. 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.

2.1.2 Stroke Renderer

In Paint Transformer, stroke is a simple 1-channel brush image, called primitive brush, which can be transformed by shape parameters and color parameters. As depicted in Figure 2.2, the shape parameters of a stroke include height h, width w; a center point coordinates x, y, and rotation angle θ. Color parameters of a stroke contain RGB values represented as r, g, and b. So, 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. As indicated in Figure 2.2, 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 stroke rendering operation:

$$I_{mid}^i = \alpha^i \cdot \bar{I}_b^i + \left(1 - \alpha^i\right) \cdot I_{mid}^{i-1} \tag{2.3}$$

and the whole *Stroke Renderer* process as:

$$I_{out} = StrokeRenderer(I_{in}, S) \tag{2.4}$$

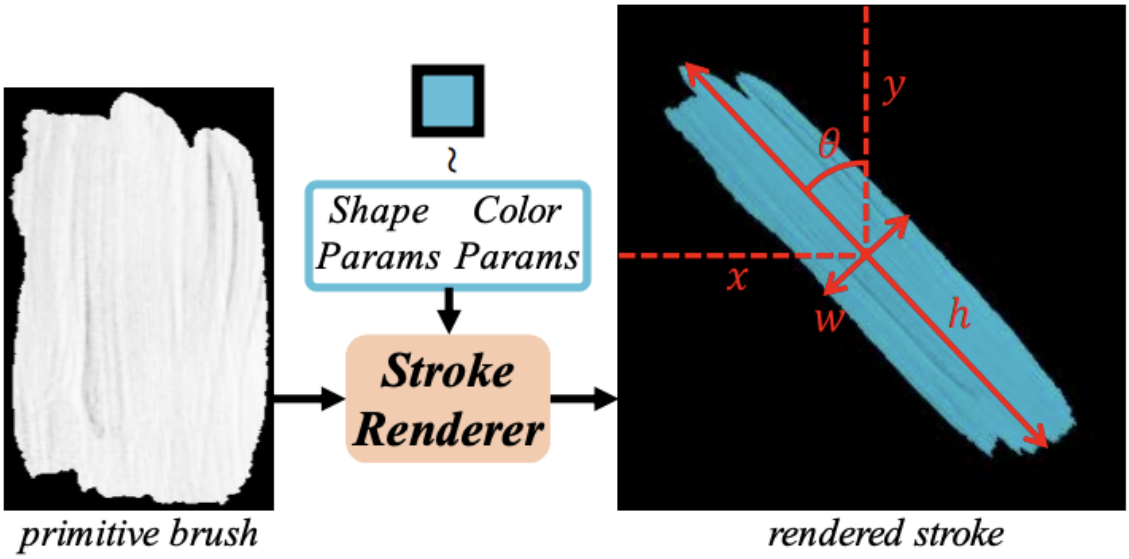


FIGURE 2.2: *Stroke Renderer* and the stroke definition (Liu et al., 2021, p. 4)

2.1.3 Stroke Predictor

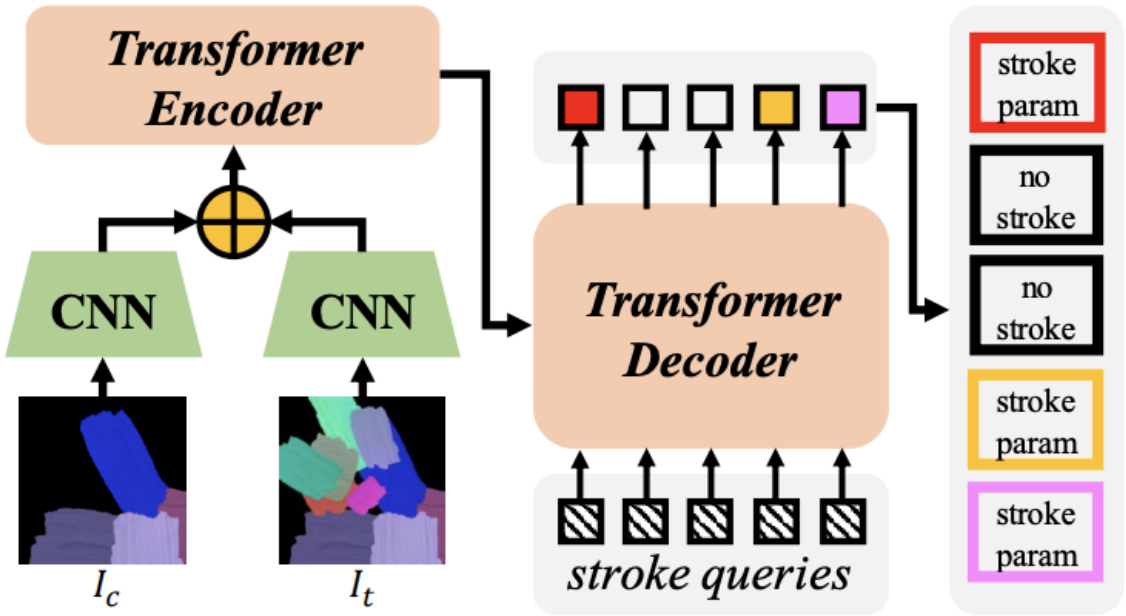


FIGURE 2.3: Overview of the *Stroke Predictor* (Liu et al., 2021, p. 4)

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. As demonstrated in Figure 2.3, 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 (Vaswani et al., 2017, p. 5) are concatenated ($\bigoplus$ in Figure 2.3) 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 (Liu et al., 2021, p. 4):

$$\frac{\partial d_i}{\partial c_i} = \frac{\partial \sigma(c_i)}{\partial c_i} = \frac{\exp(-c_i)}{(1+\exp(-c_i))^2} \tag{2.5}$$

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:

$$S_r = StrokePredictor(I_c, I_t) \tag{2.6}$$

In practice, as stated in Liu et al., 2021, p. 7, the following parameters are set to train the *Stroke Predictor*: the size of input images m as 32, and the number of strokes N as 8. The CNNs for feature extraction consists of three Convolution-BatchNormalization-ReLU blocks with two $\frac{1}{2}$-scale downsampling operations. For the Transformer, the feature size is 256, and both the encoder and decoder have three layers. N target stroke parameters during training from a uniform distribution are randomly sampled from the uniform distribution. An Adam optimizer is used with a learning rate of 0.0001. The model is trained for 30000 iterations with a batch size of 128 on a single Nvidia RTX 2080 Ti GPU, and the training time is about 4 hours.

2.1.4 Loss Function

Paint Transformer can simultaneously minimize the differences between target and prediction on both image and stroke levels. Here is the list of losses that are used to train Paint Transformer (Liu et al., 2021, pp. 4-5):

- **Pixel loss** is straightforwardly used to recreate a target image. Thus, pixel-wise loss $\mathscr{L}_{pixel}$ between I_r and I_t is minimized on the whole image level:

$$\mathscr{L}_{pixel} = \|I_r - I_t\|_1 \tag{2.7}$$

- **Stroke loss** is essential to define suitable metric for estimating the difference between strokes on the stroke level. To achieve best results, three-component losses are combined in the loss:

  - **Stroke $\mathbf{L_1}$ distance** is intuitive (s_u and s_v indicate parameters of strokes u and v, respectively):

$$\mathscr{D}_{L_1}^{u,v} = \|s_u - s_v\|_1 \tag{2.8}$$

  - **Wasserstein distance** is used because simply utilizing the L_1 metric 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

Yang et al., 2021, p. 5:

$$\begin{aligned}
\mu &= (x, y), \\
\Sigma^{\frac{1}{2}} &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \frac{w}{2} & 0 \\ 0 & \frac{h}{2} \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \\
&= \begin{bmatrix} \frac{w}{2}\cos^2\theta + \frac{h}{2}\sin^2\theta & \frac{w-h}{2}\cos\theta\sin\theta \\ \frac{w-h}{2}\cos\theta\sin\theta & \frac{w}{2}\sin^2\theta + \frac{h}{2}\cos^2\theta \end{bmatrix}
\end{aligned} \tag{2.9}$$

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

$$\mathscr{D}_W^{u,v} = \|\mu_u - \mu_v\|_2^2 + \mathrm{Tr}\left(\Sigma_u + \Sigma_v - 2\left(\Sigma_u^{\frac{1}{2}}\Sigma_v\Sigma_u^{\frac{1}{2}}\right)^{\frac{1}{2}}\right) \tag{2.10}$$

– **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:

$$\mathscr{D}_{bce}^{u,v} = -g_v \cdot \log\sigma(c_u) - (1 - g_v) \cdot \log(1 - \sigma(c_u)) \tag{2.11}$$

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 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):

$$M_{u,v} = g_v(\mathscr{D}_{L_1}^{u,v} + \mathscr{D}_{W}^{u,v} + \mathscr{D}_{bce}^{u,v}) \tag{2.12}$$

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 could be shown as:

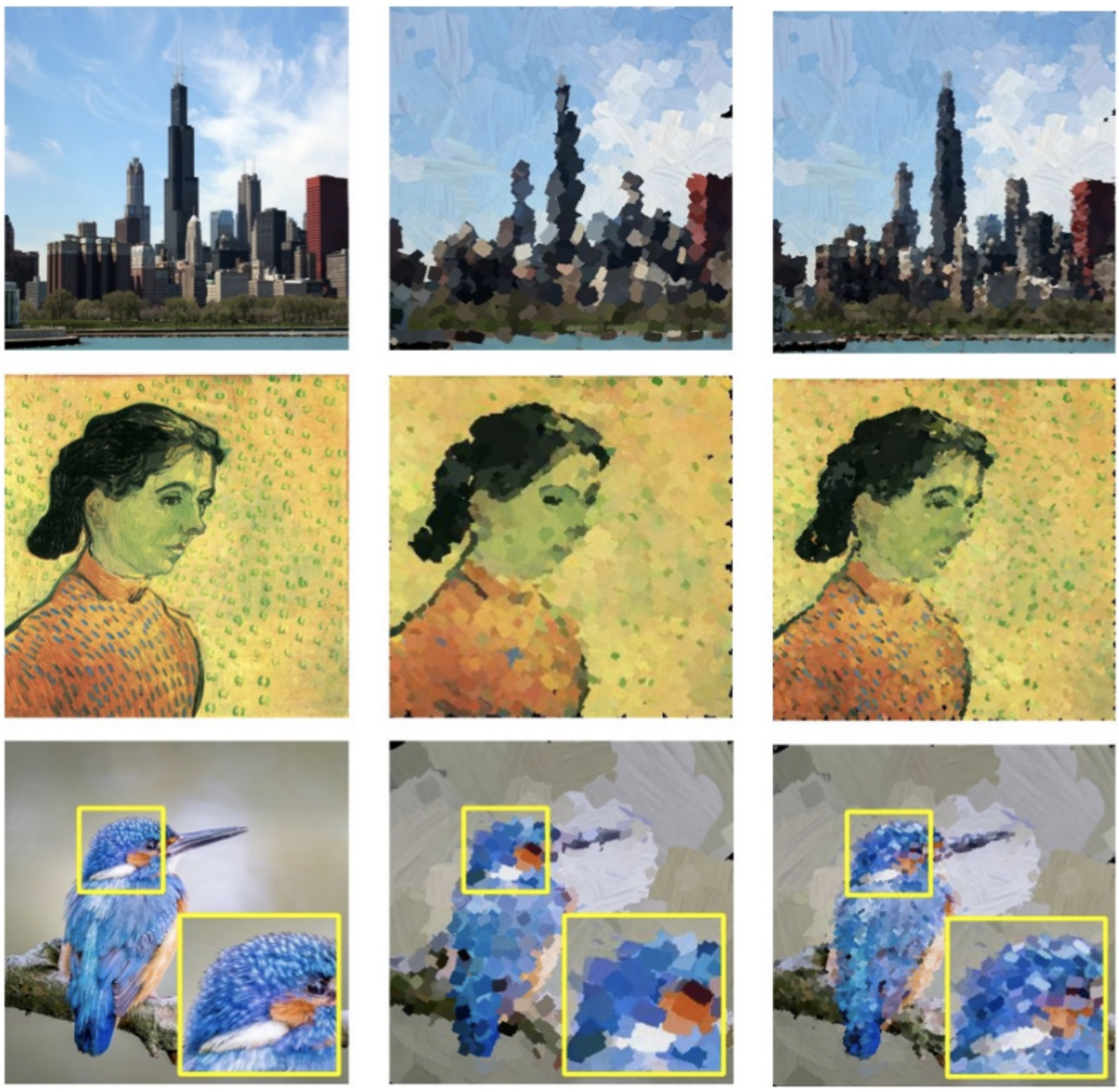
$$\mathscr{L}_{stroke} = \frac{1}{n}\sum_{i=1}^{n} g_{Y_i}(\mathscr{D}_{L_1}^{X_iY_i} + \mathscr{D}_{W}^{X_iY_i} + \mathscr{D}_{bce}^{X_iY_i}) \tag{2.13}$$

2.1.5 Inference & Results

Paint Transformer imitates a coarse-to-fine algorithm to mimic an artist to 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*. K is set as follows (Liu et al., 2021, p. 5):

$$K = \max(\text{argmin}_K(\{P \times 2^K) \geq \max(H,W)\}, 0)) \tag{2.14}$$

In the k-th $(0 \leq k \leq K)$ ranking, there are $2^k \times 2^k$ patches. Each patch is passed independently through the *Stroke Predictor* and the *Stroke Renderer*. The painting output on each ranking is derived by merging canvas patches. The following results are obtained by taking an image as an input (2.4a) with $K = 3$ output (2.4b), and $K = 4$ output (2.4c).



(A) Input image (B) Output $(K = 3)$ (C) Output $(K = 4)$

FIGURE 2.4: Paint Transformer inference results (Liu et al., 2021, p. 6)

2.2 Neural Painters: A learned differentiable constraint for generating brushstrokes

2.2.1 Overview

The paper by Reiichiro Nakano investigates different experimentations 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 (Ganin et al., 2018, p. 5) 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 determinate (Nakano, 2019, p. 2). The results are shown in Figure 2.5. For the purposes of this thesis, we are specifically interested in the GAN-reconstruction of a non-differentiable painting program brushstrokes.

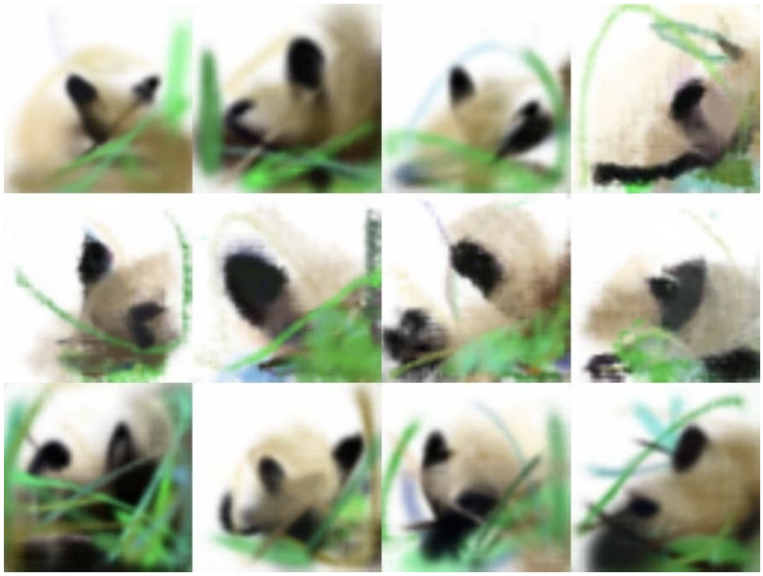


FIGURE 2.5: Neural Painters drawings of pandas (Nakano, 2019, p. 9)

2.2.2 MyPaint Brushstroke Action Space

The action space represents the set of parameters that are used as control intakes 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 (Nakano, 2019, p. 2):

- **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** consist of three variables that represent the RGB color of the brush and specified as $\{r, g, b\}$.

2.2.3 GAN Neural Painter

To recreate a MyPaint brushstroke using a neural network Nakano, 2019 proposes VAE and GAN methods. We focus here on the GAN (Goodfellow et al., 2014) 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 has its map straight 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 (Mirza & Osindero, 2014). The training operation using Wasserstein loss (Arjovsky et al., 2017) is shown in Figure 2.6. The outcomes of the training are illustrated in Figure 2.7. Pairs of true strokes on the left and the complementary GAN neural painter results on the right.

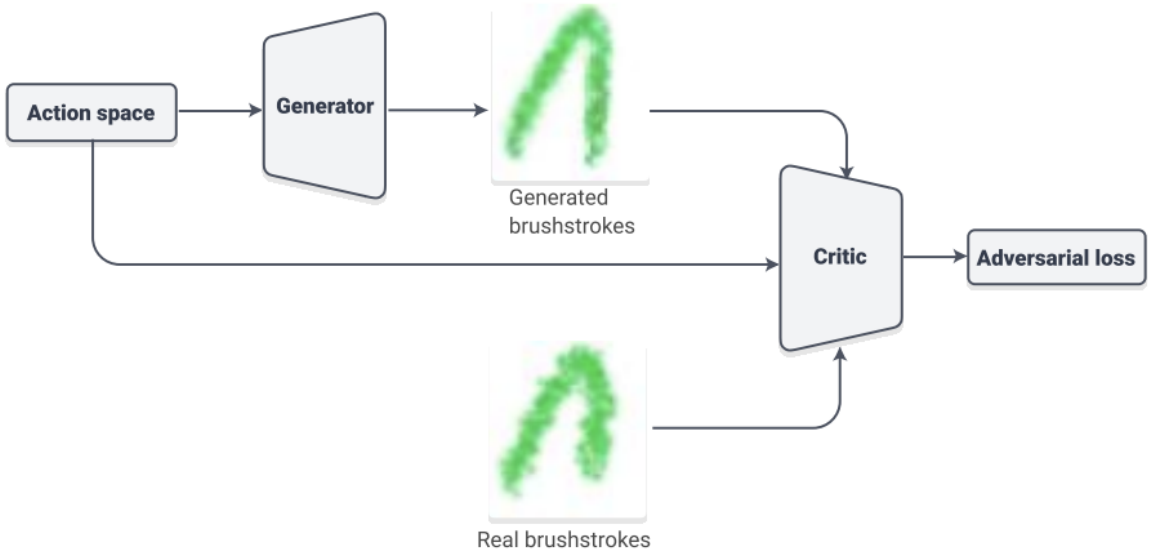


FIGURE 2.6: GAN neural painter training (Nakano, 2019, p. 4)

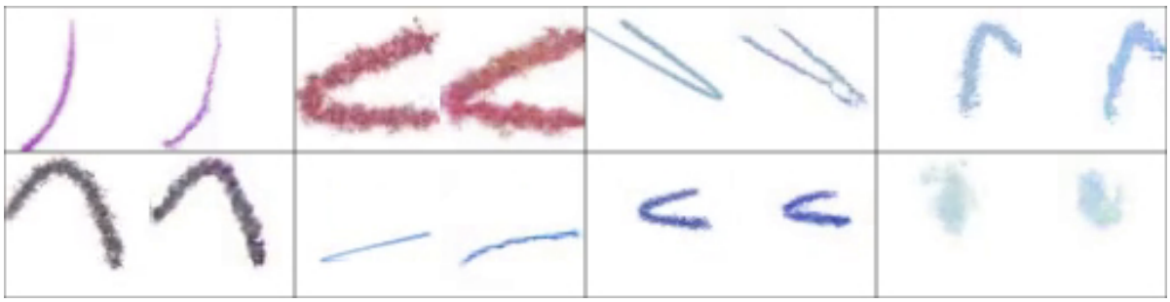


FIGURE 2.7: GAN neural painter results (Nakano, 2019, p. 4)

3 GAN-generated strokes extension for Paint Transformer

3.1 An idea to increase the complexity of Paint Transformer strokes

Paint Transformer authors state (Liu et al., 2021, p. 8) that, in their future work, it would be a worthwhile subject to study more complex strokes with different shapes or color patterns. Refined stroke rendering methods should be required, and this can additionally enhance the painting quality. In this thesis, we propose combining GAN-stroke rendering system referenced in Section 2.2.3 with Paint Transformer to introduce more complex strokes. Current Paint Transformer *Stroke Renderer* has only 8 parameters, while GAN-stroke rendering has 12 with potential to increase up to 50 parameters that MyPaint supports.

3.2 Training the GAN-generated strokes

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 (Poliakov, 2022). The training of the GAN network was done locally on a 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 Section 2.2.3 by Nakano, 2019. The modification is that we directly feed generated MyPaint strokes into the discriminator in real-time from

a data loader. At the same time, Nakano, 2019 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 (Poliakov, 2022) for convenience. The following stroke parameters are used: $\{x_s, y_s, x_e, y_e, x_c, y_c, s, p_s, p_e\}$ according to Section 2.2.2. 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 because of the CPU bottleneck.

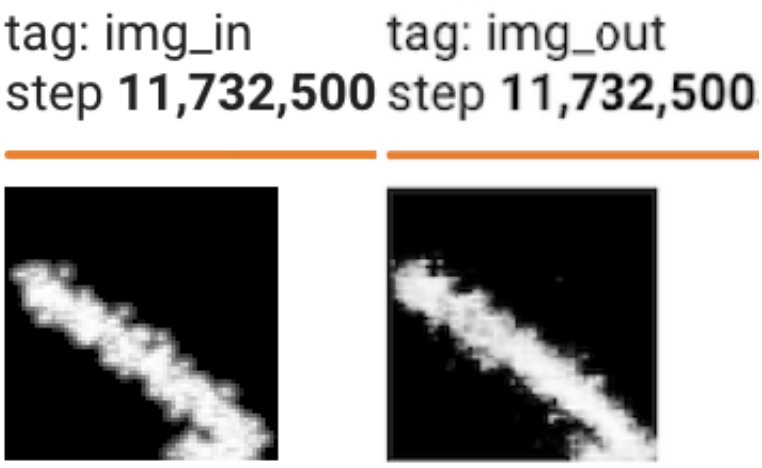


FIGURE 3.1: A GAN result sample on 11.7 million iterations

The sample of the result on the final iteration is provided in Figure 3.1. 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, generated, and discriminator scores are provided in Figures 3.2, 3.3, and 3.4, respectively. The x-axis depicts iterations, and the y-axis is the numeric score.

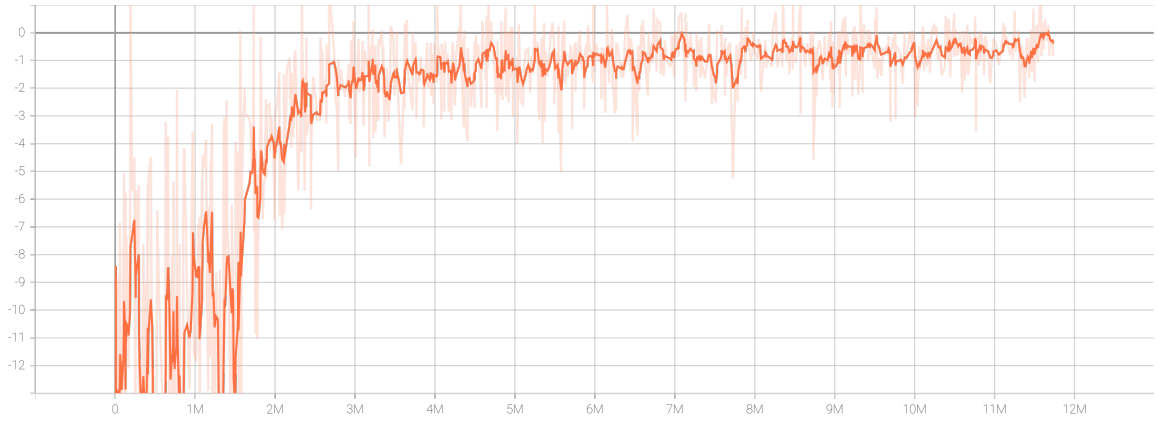


FIGURE 3.2: GAN-generated strokes discriminator loss

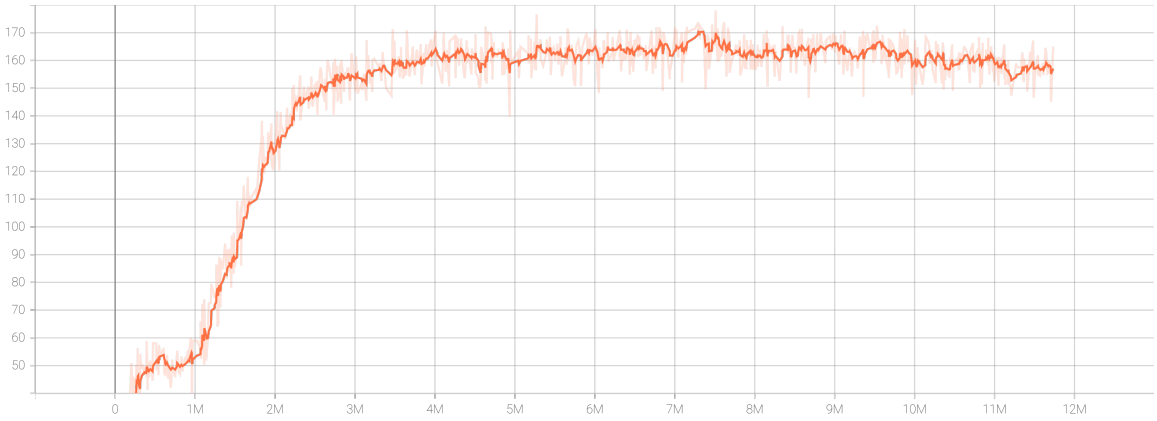


FIGURE 3.3: GAN-generated strokes generator score

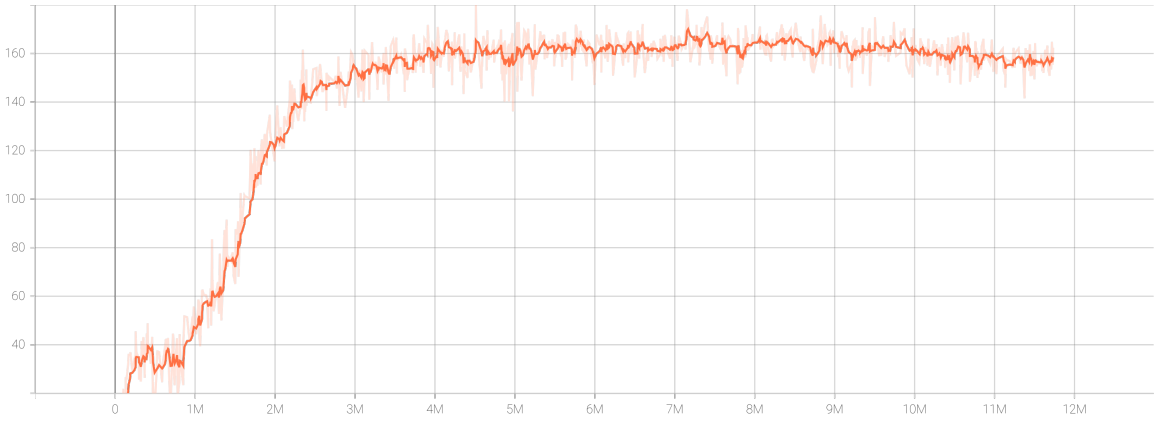


FIGURE 3.4: GAN-generated strokes real score

3.3 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 in Section 3.2 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*.

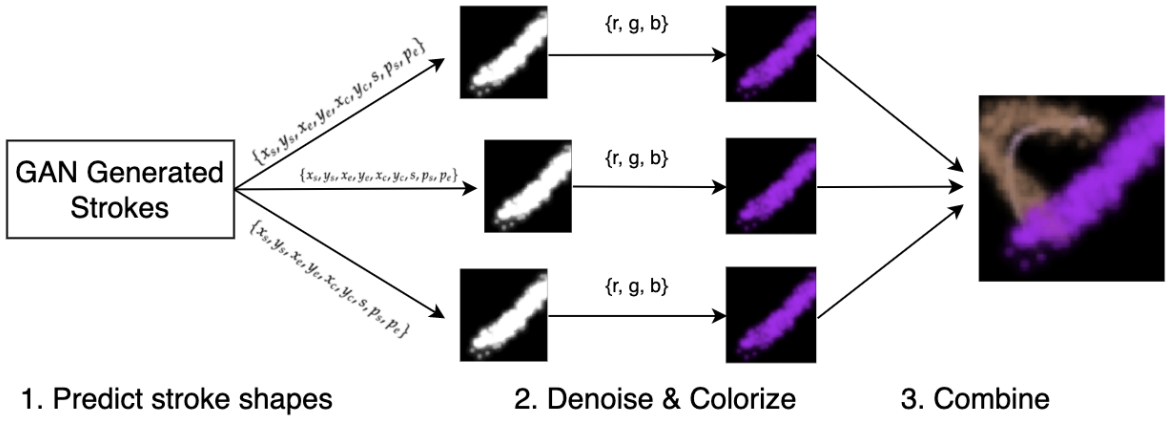


FIGURE 3.5: *GAN Stroke Renderer*

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 3.5). Note that we don't train GAN-generated strokes anymore and use them as a predictor. The GAN output does not have clear zero pixes and has numbers close to zero instead. Therefore, we cannot create a binary mask immediately and must utilize a denoising solution. Considering Equation 2.3 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 2.3 as (steps 2, 3 denoted in Figure 3.5):

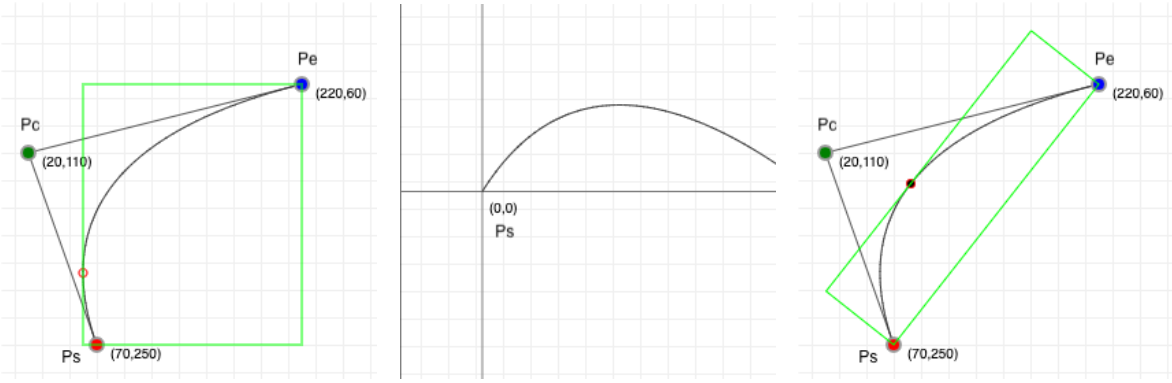
$$I_{mid}^i = \alpha^i \cdot \bar{I}_b^i \cdot c^i + \left(1 - \alpha^i\right) \cdot I_{mid}^{i-1} \tag{3.1}$$

3.4 Modifying the Stroke Predictor

Now 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 params 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 2.9). Initially, 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 $t \in [0, 1]$:

$$\begin{aligned}\mathbf{B}(t) = \sum_{i=0}^{2} B_i^2(t) \cdot P_i = \sum_{i=0}^{2} t^i (1-t)^{2-i} \cdot P_i = (1-t)^2 \cdot P_0 + 2t(1-t) \cdot P_1 + t^2 \cdot P_2 \\ = (1-t)^2 \cdot P_s + 2t(1-t) \cdot P_c + t^2 \cdot P_e \end{aligned} \tag{3.2}$$



(A) Non-rotating bounding box (B) Curve alignment (C) Rotating bounding box

FIGURE 3.6: Bezier curve (Pomax, 2015, para. 18-20)

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 $\mathbf{B}'(t) = 0$ (Pomax, 2015, para. 18):

$$\mathbf{B}'(t) = 2(1-t)(P_c - P_s) + 2t(P_e - P_c) = 0 \implies t = \frac{P_s - P_c}{-2 \cdot P_c + P_s + P_e} \tag{3.3}$$

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 3.6a). To get a rotated bounding box, we need to make $P_s = (0,0)$ and align the curve on the x-axis via (Figure 3.6b):

$$\begin{aligned}
&\alpha = \arctan\frac{y_e}{x_e} \implies R = \begin{bmatrix} \cos(-\alpha) & -\sin(-\alpha) \\ \sin(-\alpha) & \cos(-\alpha) \end{bmatrix} \\
&\dot{P}_s = P_s - P_s = (0,0) \\
&\dot{P}_c = (P_c - P_s) \cdot R \\
&\dot{P}_e = (P_e - P_s) \cdot R
\end{aligned} \tag{3.4}$$

Afterward, we calculate a non-rotating bounding box for $\dot{P}_s, \dot{P}_c, \dot{P}_e$ via Equations 3.2, 3.3 and make a reverse transformation (Pomax, 2015, para. 19-20):

$$\begin{aligned}
\dot{P}_{min} &= min(\dot{\mathbf{B}}(t), \dot{P}_s, \dot{P}_e) = (\dot{x}_{min}, \dot{y}_{min}) \\
\dot{P}_{max} &= max(\dot{\mathbf{B}}(t), \dot{P}_s, \dot{P}_e) = (\dot{x}_{max}, \dot{y}_{max}) \\
(x_{max}, y_{max}) &= (\dot{x}_{max}, \dot{y}_{max}) \cdot R^{-1} + P_s \\
(x_{min}, y_{min}) &= (\dot{x}_{min}, \dot{y}_{min}) \cdot R^{-1} + P_s \\
(x'_{max}, y'_{max}) &= (\dot{x}_{max}, \dot{y}_{min}) \cdot R^{-1} + P_s \\
(x'_{min}, y'_{min}) &= (\dot{x}_{min}, \dot{y}_{max}) \cdot R^{-1} + P_s
\end{aligned} \tag{3.5}$$

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 3.6c.

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):

$$\begin{aligned} x_s &= \text{clamp}(x_s + 0.15 \cdot p_s,\, 0,\, 1);\;\; y_s = \text{clamp}(y_s + 0.15 \cdot p_s,\, 0,\, 1) \\ x_e &= \text{clamp}(x_e + 0.15 \cdot p_e,\, 0,\, 1);\;\; y_e = \text{clamp}(y_s + 0.15 \cdot p_e,\, 0,\, 1) \\ x_c &= \text{clamp}(x_c - 0.15 \cdot s,\, 0,\, 1);\;\; y_c = \text{clamp}(y_c - 0.15 \cdot s,\, 0,\, 1) \end{aligned} \tag{3.6}$$

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

$$\begin{aligned} x &= \frac{x_{max} + x_{min}}{2}; \quad y = \frac{y'_{max} + y_{min}}{2} \\ w &= \sqrt{(y'_{max} - y_{max})^2 + (x'_{min} - x_{max})^2} \\ h &= \sqrt{(y_{max} - y'_{min})^2 + (x_{max} - x'_{max})^2} \\ \theta &= \arctan \frac{y'_{max} - y_{max}}{x'_{max} - x_{max}} \end{aligned} \tag{3.7}$$

The code implementation can be found in `get_rotated_bounding_box` method, and the resulting bounding boxes on GAN-generated strokes are shown in Figure 3.7.

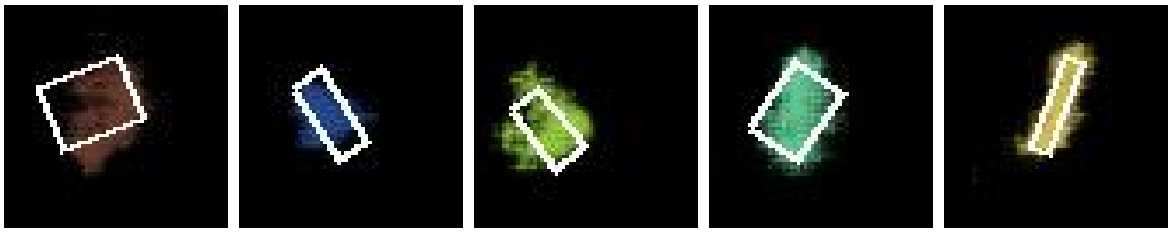


FIGURE 3.7: Rotating bounding box on GAN-generated strokes

3.5 Training the GAN-generated strokes with Paint Transformer

Once the *Stroke Predictor* is optimized for the new parameters set, 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 3.8, and 3.9, 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 3.10, 3.11, respectively. The x-axis depicts iterations, and the y-axis is the numeric score. Canvas-target-predict triads S_b, S_f, and S_r, referenced in Section 2.1.1, are shown in Figure 3.12 during the training. We also changed the monitoring framework from Visdom to a commonly used Tensorboard.

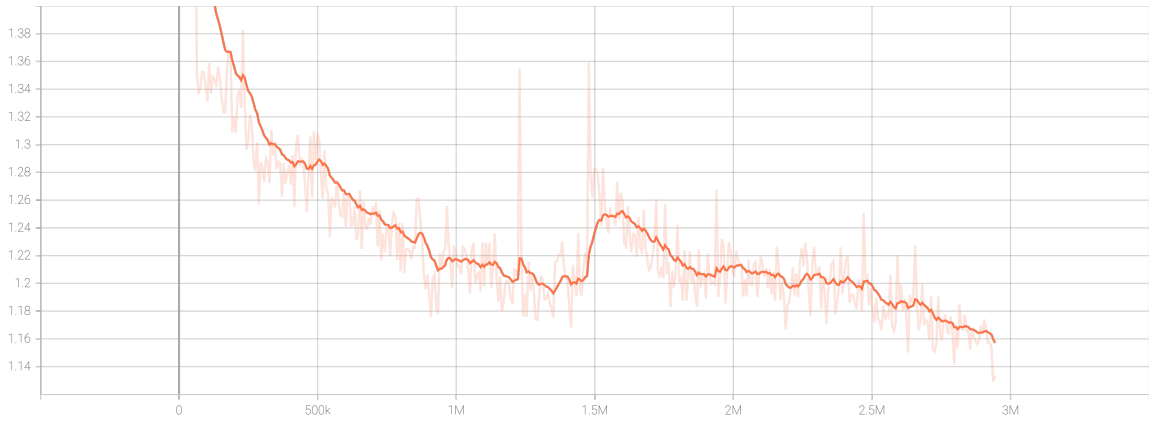


FIGURE 3.8: Paint Transformer pixel loss

FIGURE 3.9: Paint Transformer stroke L_1 distance loss

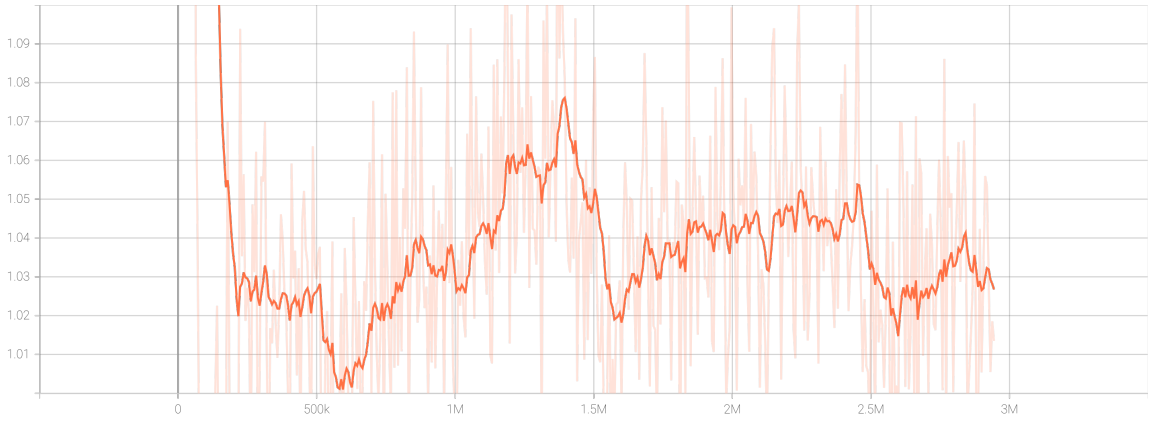


FIGURE 3.10: Paint Transformer Wasserstein distance loss

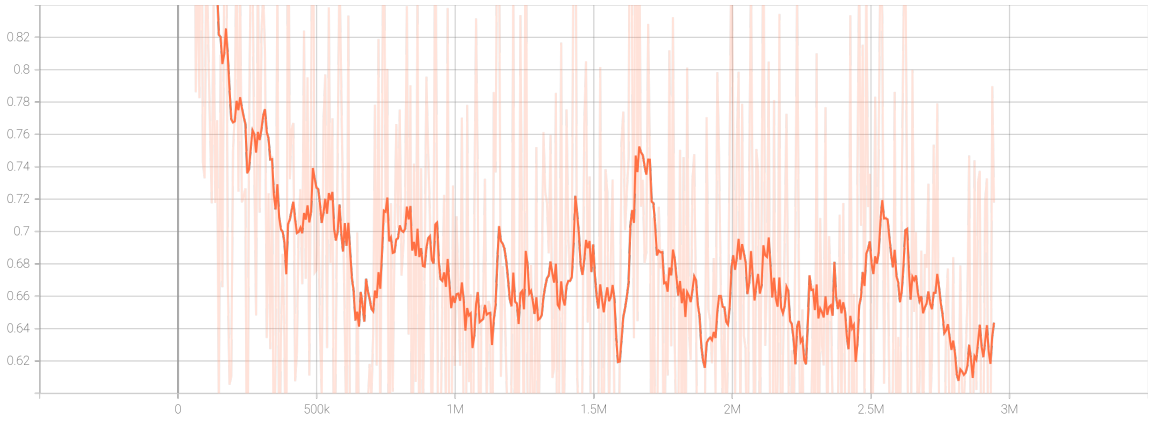


FIGURE 3.11: Paint Transformer binary cross-entropy decision loss

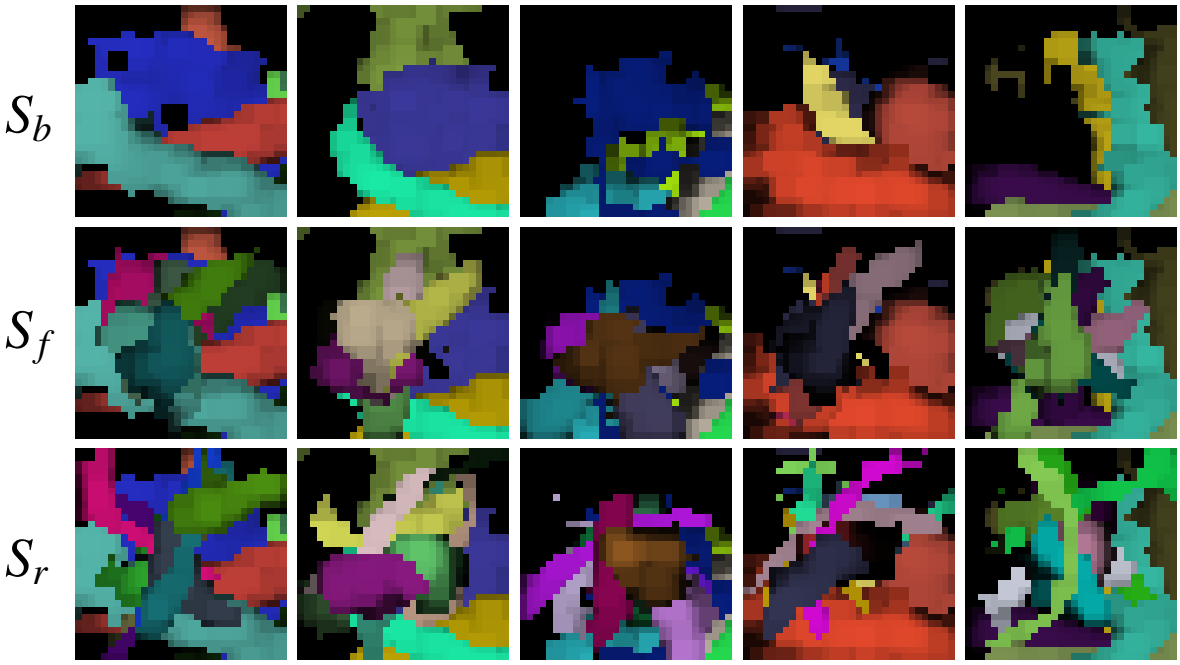
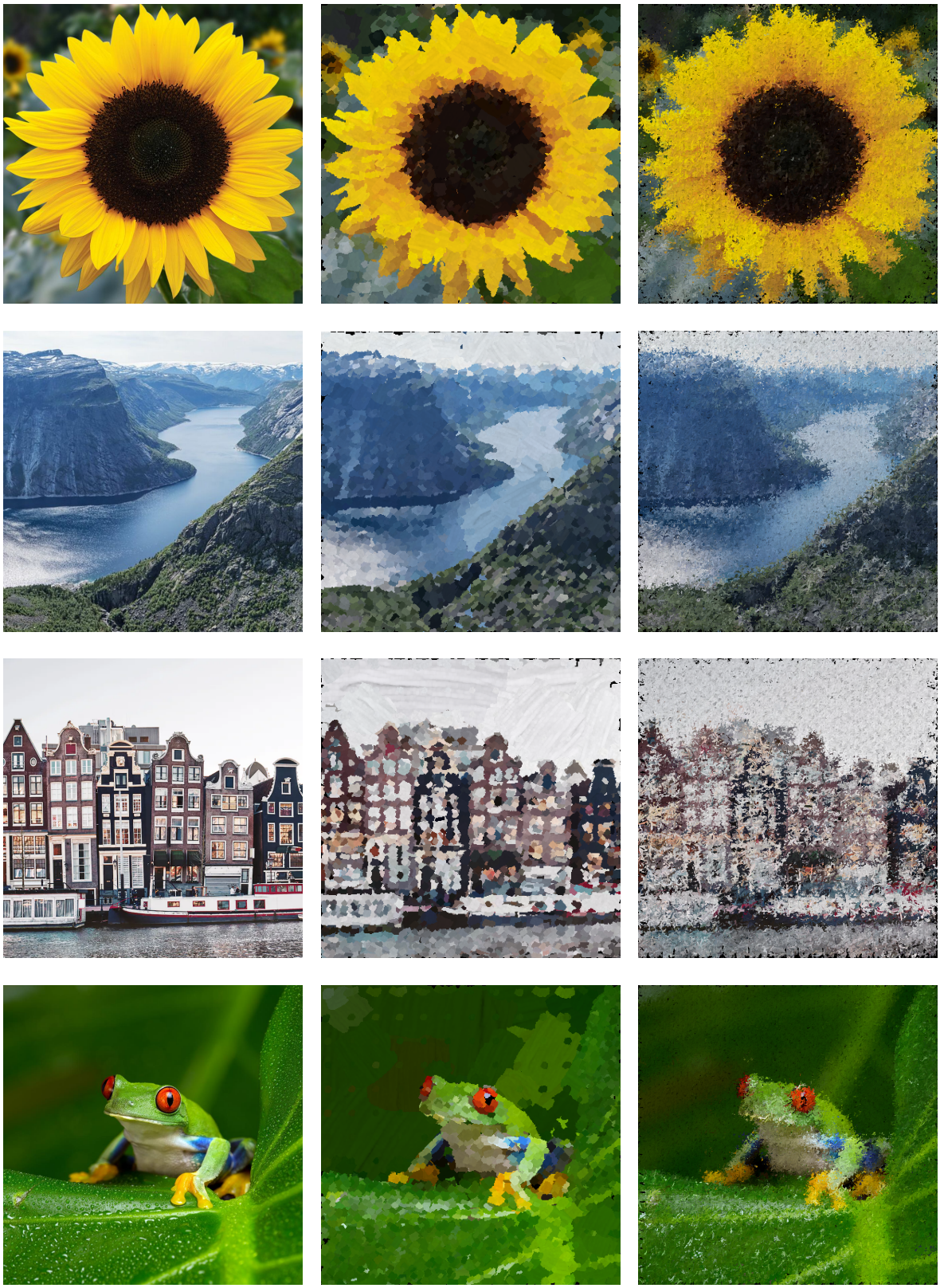


FIGURE 3.12: Canvas-target-predict triads in training

3.6 Results

We modified the inference module so it could work with *GAN Stroke Renderer* and obtained the results depicted in Figure 3.13c. We take Figure 3.13a as an input, and we also show results obtained by the original Paint Transformer in Figure 3.13b. 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 (we have the same K as the original). This creates a more abstract painting style, especially seen in the fjord, city, and sunflower cases. We can also notice that the Paint Transformer extension does not paint larger strokes for the uniform patches. We can also notice that the Paint Transformer extension does not paint larger strokes for the uniform patches. We need to modify further Wasserstein distance loss and a binary cross-entropy decision loss to improve the results.



(A) Input image (B) Original output (C) GAN-extension output

FIGURE 3.13: Results

4 Conclusion

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 8 original. We partly modified the loss function to accept a new parameter list. The results we got have a different painting style and are more abstract; however, the extension paints strokes in a similar size. This indicates that we need to make further effort in modifying Wasserstein distance loss and a 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 result by removing artifacts on the generated strokes.

Bibliography

Elgammal, A., Liu, B., Elhoseiny, M., & Mazzone, M. (2017). Can: Creative adversarial networks, generating "art" by learning about styles and deviating from style norms. https://doi.org/10.48550/ARXIV.1706.07068

Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. https://doi.org/10.48550/ARXIV.1508.06576

Haeberli, P. Paint by numbers: Abstract image representations. In: *Proceedings of the 17th annual conference on computer graphics and interactive techniques*. SIGGRAPH '90. Dallas, TX, USA: Association for Computing Machinery, 1990, 207–214. ISBN: 0897913442. https://doi.org/10.1145/97879.97902.

Ha, D., & Eck, D. (2017). A neural representation of sketch drawings. https://doi.org/10.48550/ARXIV.1704.03477

Zhou, T., Fang, C., Wang, Z., Yang, J., Kim, B., Chen, Z., Brandt, J., & Terzopoulos, D. (2018). Learning to sketch with deep q networks and demonstrated strokes. https://doi.org/10.48550/ARXIV.1810.05977

Zou, Z., Shi, T., Qiu, S., Yuan, Y., & Shi, Z. (2020). Stylized neural painting. https://doi.org/10.48550/ARXIV.2011.08114

Liu, S., Lin, T., He, D., Li, F., Deng, R., Li, X., Ding, E., & Wang, H. (2021). Paint transformer: Feed forward neural painting with stroke prediction. https://doi.org/10.48550/ARXIV.2108.03798

Nakano, R. (2019). Neural painters: A learned differentiable constraint for generating brushstroke paintings. https://doi.org/10.48550/ARXIV.1904.08410

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. https://doi.org/10.48550/ARXIV.1706.03762

Yang, X., Yan, J., Ming, Q., Wang, W., Zhang, X., & Tian, Q. (2021). Rethinking rotated object detection with gaussian wasserstein distance loss. https://doi.org/10.48550/ARXIV.2101.11952

Ganin, Y., Kulkarni, T., Babuschkin, I., Eslami, S. M. A., & Vinyals, O. (2018). Synthesizing programs for images using reinforced adversarial learning. https://doi.org/10.48550/ARXIV.1804.01118

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. https://doi.org/10.48550/ARXIV.1406.2661

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. https://doi.org/10.48550/ARXIV.1411.1784

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. https://doi.org/10.48550/ARXIV.1701.07875

Poliakov, M. (2022). Paint-transformer-gan. https://github.com/mxpoliakov/PaintTransformerGAN

Pomax. (2015). A primer on bezier curves. https://pomax.github.io/bezierinfo