diffvg+CLIP: Generating Painting Trajectories from Text

Gerry Chen, Alice Dumay, Mengyi Tang

Abstract

Given a sentence, the goal of our project is to find out the optimized trajectories for a painting robot to paint an image that illustrates the context of the sentence. We investigate two solution approaches. In the first approach, we consider two stages: in the first stage a set of candidate images are generated in a classification task to match the given sentence, and in the second stage a candidate raster image is simplified into a robot-paint-image along with a vector trajectory with a color labels. For the first stage, we implemented CLIP (Contrastive Language-Image Pretraining), a computer vision system released by OpenAI to generate candidate images. In the second stage, we proposed two different approaches: (i) using an off-the-shelf differentiable vector graphics rasterizing library named diffvg and (ii) applying a simple image segmentation task SLIC and backpropagating through a probability-based network to achieve a set of probability measure, which can be represented as the trajectory. In the second approach, we pair diffvg with CLIP and back-propogate through the entire chain to produce vector graphics which are described by a sentence. Additional details are provided in Section 2.

1. Introduction and Related Work

Text to image generation has become popular as deep learning proposes powerful tools and advanced feature representation explored by researchers. For example, in [15], a generative adversarial text to image synthesis is proposed to produce a synthesis of realistic images from text. The authors used a GAN formulation to build a connection between text features and image pixels. In addition, [18] investigate the same problem using GAN and introduce a novel conditioning augmentation technique to impose smoothness in the latent conditioning manifold. An extension of this work can be found in [19]. More works in this field can be found in [17, 20, 14, 10].

However, all these works have their limitations when it comes to painting the image out by using a robot in the implementation since the high resolution of the output image may not be a good fit for a simple painting task in an optimized trajectory. One way to achieve a trajectory from given text is to build a bridge between the high-resolution images and vectored images. Works have been done in this area various methods. For example, in [12], an image vectorization technique that operates on a color image augmented with a depth map is proposed to use both color and depth edges to define a vectorized path. In [11], the author first proposed a new form of vector graphics Temporal Diffusion Curve (TDC) to model the evolution of strokes. In [7], Posson Vector graphics (PVG) is presented as an extension of the popular diffusion curves to generate smooth-shaded images. In [3], the authors encoded global manipulation geometries and local image details within a hybrid vector structure, using parametric patches and detailed features for localized and parallelized thin-plate spline interpolation. However, all these works have a complex representation with a heavy computation time of the vectorized images and do not guarantee that the trajectory is optimized for the painting task. For this reason, we want to design a simpler way to find out a suitable trajectory for real painting jobs associated with the text-image task.

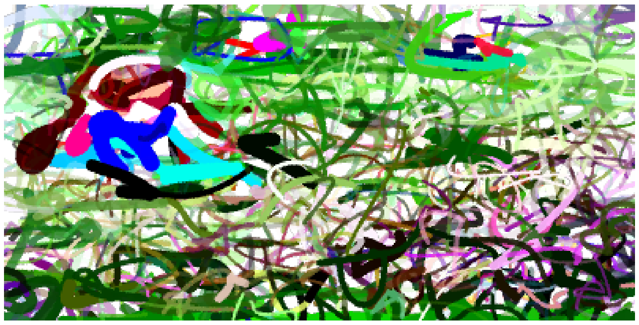


Figure 1. Our algorithm inputs a text prompt ("**A small girl in the grass plays.**") and outputs a robot trajectory which, when executed, will paint a picture. This image is the simulated painting that would result from a robot following the trajectory generated by the prompt.

Motivated by the task of painting using a robot (i.e. generating trajectories that will produce quality paintings), we present an application of CLIP and diffvq to perform the task of *say-and-paint*. We pursue two different approaches: (1) first generating a reference raster image then vectorizing it and (2) directly optimizing a vector representation by back-propagating through a differentiable rasterizer feeding into CLIP. In both cases, the vector representation equates to a robot trajectory, where the xy trajectory corresponds to the robot path and the stroke width corresponds to the distance of the robot spray paint to the canvas. See [2] for an example robot platform created by one of us authors that can execute such trajectories with real paint.

The first proposed method has two stages: in the first stage, we use CLIP to generate a raster image given a sentence. In the second stage, we use a differentiable rasterizer to optimize for a vectorized image which, when rasterized, will best match the target raster image. We develop our own differentiable rasterizer in which a simple image segmentation task is performed followed by a *density map* learning strategy in which the trajectory is represented by multiple Gaussian distributions at different timesteps in each subregion of a unified color of a segmented image. We also apply diffvg [9] to our task as a baseline.

The second proposed method feeds the raster output of a differentiable rasterizer as the input of CLIP, then optimizes for the vector input which generates minimizes the difference between the raster image embedding and the target text prompt embedding.

The main contributions of this work are outlined below:

1. We introduce a simple methodology (sentence – image – trajectory) to produce a vectorized image from a given sentence, where the image illustrates the content in the content of the sentence.

2. We build a simple model to learn the trajectory of a binary image and extend it to a colorful image.

3. We implemented it using CLIP and the proposed method to produce a vectorized image along with trajectories to achieve the task. We also implemented combining the existing methods CLIP and Diffvq to produce vectorized images.

4. We provide experimental results based on the proposed method and showed examples where the proposed method can give decent results.

2. Method

2.1. Stage I (Input Sentence, output image)

In this section, we generated some images illustrating a sentence given as an input. This was done based on CLIP (Contrastive Language-Image Pretraining), a computer vision system released by OpenAI [13]. This model was trained to perform zero-shot learning, which means to give significant results on any classification dataset without being trained on it previously. CLIP was trained to predict how likely a given sentence matches with an arbitrary image. This is called contrastive pre-training.

The text encoder, together with an image encoder is generating 9 images for each input sentence. To get better performances, we chose to fine-tune CLIP with the Flick30k dataset, composed of 31,000 images from Flickr, together with 5 reference sentences provided by human annotators. We build our encoders, with the neural architecture of resnet50 [6] as an image encoder, together with a distilBERT text encoder [16]. On top of that, we added projection heads. We put all these elements into CLIP and were able to generate images.

2.2. Stage II - A (Input Image, output trajectories) use SLIC to do image segmentation

In this section, we present the method to produce trajectories for a robot to paint a given image by considering the stroke as a series of Gaussian distributions in 2D where the mean and standard deviation can be understood as location and radius. To handle images of different intensities of color and reduce computation cost, we first perform an image segmentation task then minimize a loss of the total distribution of the output image. Our goal is to produce a paint that is as close as possible to the segmentation result and minimize the total length of the trajectory. The detail of the proposed method is as follows:

For a given image I, we first perform an unsupervised image segmentation task by using `SLIC` [1, 8] which adapts the traditional k-means method in clustering approach to efficiently generate super-pixels. A segmented image then can be represented by

$$\tilde{I} = k \ (0 \le k \le K-1). \tag{1}$$

Here K is the total number of segments in the segmentation task. Notice that K can also be predetermined in the supervised task and this can be dependent on the complexity of a given image. For the best result of visualization, we suggest K be between 20 and 70.

For simplicity, let's denote the region where $\tilde{I} = k$ to be Ω_k such that

$$\tilde{I} = \cup_k \Omega_k. \tag{2}$$

Hence in each subs-task, we consider our target image to be

$$I^k_{target} = \begin{cases} 1 & \tilde{I} = k \\ 0 & o.w. \end{cases} \tag{3}$$

for $0 \le k \le K-1$.

For each k $(0 \le k \le K-1)$, we then initialize a sequence $\{\boldsymbol{v}^k_i\}$, where each element $\boldsymbol{v_i}$ denotes the an independent 2D Gaussian distribution with mean x^k_i, y^k_i and standard deviation σ^k_i, σ^k_i:

$$\boldsymbol{v}^k_i = (x^k_i, y^k_i, \sigma^k_i) \sim \mathcal{N}((x^k_i, y^k_i), (\sigma^k_i, \sigma^k_i)) \tag{4}$$

$$0 \le k \le K-1, 0 \le i \le N-1$$

where K is the total number of segments of a given image and N is the total number of discrete points in each trajectory. For simplicity we consider $N = 8$ for all segments. This is based on the assumption that the given image is smooth enough to be segmented into images of similar complex sub-regions. For the case where an image has an irregular shape of boundaries with a simple shape of main objects, we suggest considering different N for each k where a complexity measurement can be designed as an extension of our work.

For faster convergence, we consider the initial condition to be

$$(x_i^k, y_i^k, \sigma_i^k) = (x_0^k + r\cos(\frac{2\pi i}{N}), y_0^k + r\sin(\frac{2\pi i}{N}), c + \frac{i}{N}), \tag{5}$$

where c_1, c_2, c_3 are predetermined coefficient and (x_i^k, y_i^k) denotes the *center* of the sub-region Ω_k where $\tilde{I}_{ij} = k$. Here we consider $r = 10, c = 2$ as default values. Notice that in the initial condition, the x_i, y_i are required to be in the shape of mincing the trajectory instead of random walk for faster convergence.

Now we can further represent the output image with respect to Ω_k to be

$$I_{out}^k = 2\left[1 - \frac{1}{1 + \exp - \sum_{i=0}^{N} \mathcal{N}((x_i^j, y_i^j), (\sigma_i^j, \sigma_i^j))}\right] \tag{6}$$

Figure 2 shows an example of where the sub-region Ω_k is attached to the boundary condition. Without the *pushing back* restriction in Equation (6), it's possible that the trajectories may go to the other side of the image under periodic boundary conditions. In order to guarantee that the means of distribution (x_i^j, y_i^j) is located inside of region Ω_k, we consider

$$x_i^j = \min(\max(0, x_i^j), N_x - 1), y_i^j = \min(\max(0, y_i^j), N_y - 1) \tag{7}$$

where N_x, N_y are the size of the normalized image. In our experiments, we set $(N_x, N_y) = 128 \times 128$.

To let the sequence $\boldsymbol{v}_i^j$ converge to the trajectory that produces a binary image I_{target}^k close to the target image, we design our loss function motivated by the two following scenarios:

1. the trajectory is possible to produce the target image while σ_i^j is not large enough to cover the target area to too large to over paint the unnecessary area. For this reason, we want to minimize

$$||I_{out}^k - I_{target}^k||_2$$

2. The target image is close to the output image while the locations (x_i, y_i) are scattered, "discontinuous", and/or revisiting old locations. For this reason, we want to minimize

$$\sum_{i=0}^{N-1} ||\boldsymbol{v}_{i+1}^k - \boldsymbol{v}_i^k||_2.$$

Now we represent our loss function as follows:

$$Loss = \alpha||I_{out} - I_{target}||_2 + \beta \sum_{i=0}^{N-1} ||\boldsymbol{v}_{i+1}^j - \boldsymbol{v}_i^j||_2. \tag{8}$$

Assuming that the initial condition is large enough to cover the target, we consider more heavily penalizing the region where is over-painted. Hence we further represent our loss function as follows:

$$\begin{aligned} Loss =& \alpha||2ReLu(I_{out} - I_{target}) + ReLu(I_{target} - I_{out})||_2 \\ &+ \beta \sum_{i=0}^{N-1} ||\boldsymbol{v}_{i+1}^j - \boldsymbol{v}_i^j||_2. \end{aligned} \tag{9}$$

Then we apply Stochastic Gradient Descent on each target image I_{target}^k to obtain the painting trajectory in this region Ω_k to be:

$$I_{paint}^k = (k_R, k_G, k_B) I_{out}^k \in \mathbb{R}^{N_x \times N_y \times 3}, \tag{10}$$

where (k_R, k_g, k_b) denotes color intensity of the segmented image in sub-region Ω_k for $0 \le k \le K$. Figure 3 shows an example of how the trajectory converges to the initial a sub-region from the initial condition. The last column shows the comparison between the target binary image(in blue) and the optimized output of the paint w.r.t the subregion(in pink). It is shown that after 13 iterations, the over-painted region is shrunk and the total region converges close to the target. After 39 iterations, the general shape of the target image is achieved. After 52 iterations the delicate boundaries are showing out more.

Notice that there exists some small region near the boundary of Ω_k that is over-painted. For this reason, we apply a mask $\mathbb{I}(\Omega_k)$ which denotes the characteristic equation on ω_k to deal with this issue. Finally, we represent the whole paint by trajectory as follows:

$$I_{paint} = \sum_{k=0}^{K} I_{paint}^k \mathbb{I}(\Omega_k) \tag{11}$$

2.3. Stage II - B (Input Image, output trajectories) Use Diffvg

As an alternative to our custom paint rendering optimization using Gaussian distributions, we also produce trajectories of the same images using an off-the-shelf differentiable vector rendering library called diffvg. While our renderer inputs a set of locations and parameters for Gaussian

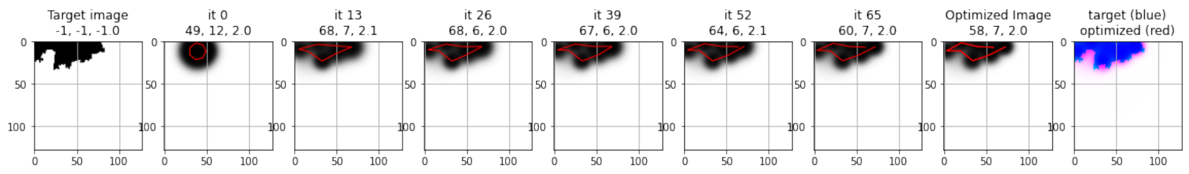


Figure 2. An example of converging to a sub-region Ω_k attached to the boundary of the original image.

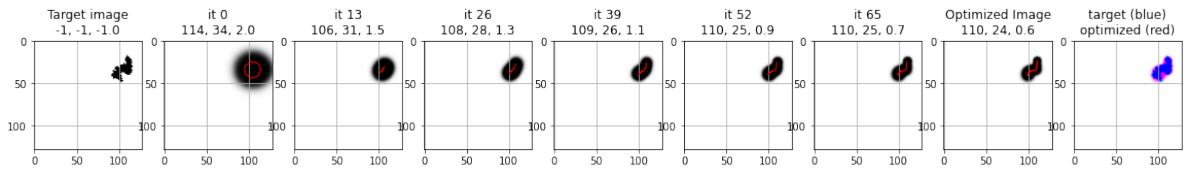


Figure 3. An example of converging to a sub-region from an over-painted initial condition after 80 steps.

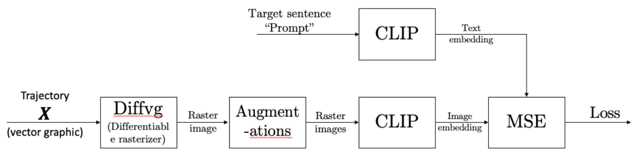


Figure 4. Our "Stage I and II combined" algorithm to generate vector trajectories given a text prompt

marks, the diffvg work inputs a set of continuous (polynomial) strokes. Although diffvg has the drawback of neglecting the non-uniform paint distribution inherent to spray paint dispersion, it is more mature than our Gaussian-based approach. diffvg uses the shortest distance from pixel to curve to form a gradient and employs sophisticated methods to differentiably compute shortest distances and handle anti-aliasing (since aliasing otherwise causes mis-behaved gradients).

A similar optimization procedure is used to find the optimal set of stroke parameters to fit a raster target image by backpropagating and performing gradient descent on the stroke parameters. RMSE between the target and rendered image was used as the loss function:

$$Loss(x) = ||I_{target} - f(x)|| \tag{12}$$

where $||\cdot||$ denotes the Frobenius norm, I_{target} is the target image, $f(\cdot)$ is applying diffvg to render an image, and x are the stroke parameters. The stroke parameters are defined as a list of cubic bezier curve coefficients and widths (one width per stroke) and diffvg smoothly interpolates between xy waypoints using cubic spline interpolation.

The points, stroke widths, and colors are optimized with the Adam optimizer with learning rates of 1.0, 0.1, and 0.01 respectively for 150 iterations.

2.4. End-to-end Stage I and II combined

Finally, we attempt to omit the intermediate rasterized image step by combining the differentiable vector graphics renderer with CLIP and back-propagating through the entire network to directly optimize for a vector representation that best represents a text prompt. Figure 4 summarizes our algorithm.

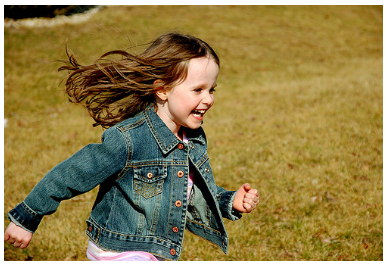


Figure 5. Image generated with the CLIP architecture and input sentence "A small girl in the grass plays"

It should be noted that, after completing our code, we discovered that CLIPDraw [5] is a pre-existing work from 2021 with a virtually identical algorithm. This is not particularly surprising, since we were inspired by the popular VQGAN+CLIP wave [4] (in which a traditional, raster-producing GAN is paired with CLIP) which undoubtedly inspired many derivative works. It should hopefully be evident from our code that we did not use CLIPDraw's implementation, however, we did borrow ideas (such as the image augmentations) from commonplace VQGAN+CLIP techniques.

3. Experimental results

3.1. Stage I

From our built network based on CLIP, we were able to input a sentence, such as *"A small girl in the grass plays"* and the model gave us images similar to the one presented in Figure 5. We noticed that the model gives better results when the sentence is more descriptive. Giving a single word as the input ended up with poorly generated images.

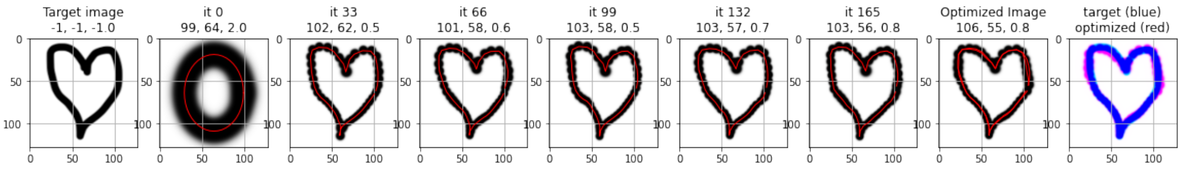


Figure 6. An example of painting unified thickness of irregular shape binary object. It is shown that after 33 iteration, the general shape of the object is found.

3.2. Stage II -A

We present our vectorization results on various tasks using our own differentiable rasterizer.

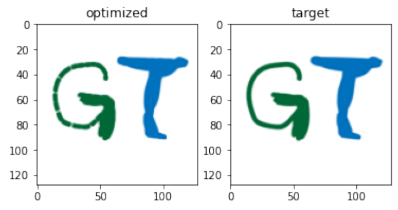
- **Task on binary image with unified background**
We started building this project by working on a binary image where only 0 and 1 are given as values of pixels. Figure 6 shows an example of converging from a circle to a heart shape. In this example, only the back regions are considered to be painted, which indicates that the trajectory should follow the curve's shape ideally. It is shown that after 33 iterations, the general shape of the object is found. This example shows that the proposed method is capable to identify the trajectory of any shape. However, a closed curved is favorable, which is also the limitation of the proposed method.

- **Task on a colorful image with disconnected region with unified background**
We then explore the method to deal with a colorful image. We started with a simple image with white background and two letters G and T with green and blue color, which is shown in the target image in Figure 7(a). In this example, we consider the original image as a binary image, then perform learn the trajectory over the two letters as a whole. In the end, we impose the color that matches the color in the original image to the stroke (Gaussian). Figure 7(b) showed the converging status of the trajectories. It is shown that the general shape is found after 249 iterations. From this example, we can extend from binary image to colorful image. However, we see the limitation that the trajectories consider painting everything as a whole in the given image while this will cause some issue when the image has more complicated, or the image has smoothly changing color, or the background is considered to be painted.

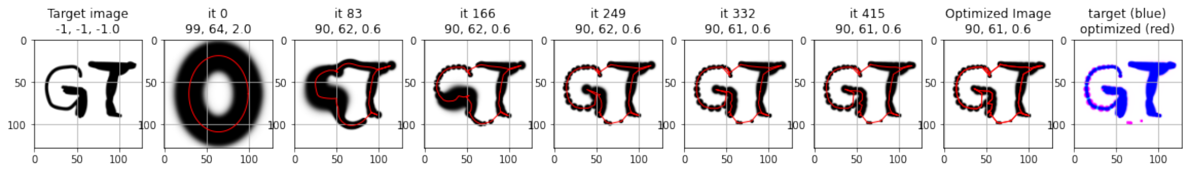
- **Performing image segmentation task to do multiple learnings on a discontinuous colorful image with background**
We further explore utilizing image segmentation task `SLIC` to segment a discontinuous image into multiple sub-regions including the background, then implement the method described in section 2.2 to perform multiple tasks. Figure 9 shows the result of an example of painting 8 sub-regions of a given region by using $N = 8$ as the total number of time steps in each sub-region. It is shown that by doing multiple tasks in each region Ω_k, the proposed method can identify the optimal trajectories in each region and paint a roughly close image.

- **The effect of time step N in each sub-region**
Figure 9 shows an example of using different numbers of time steps in painting each sub-region. It is shown that when $N = 20$, the white background is covered more by the stroke while when $N = 10$, there is some blurry black region that is not covered by the Gaussian distribution. This is because when (i) the area to paint is relatively large, (ii) the sub-region Ω_k's shape is complicated enough such that the radius σ_i^j can not be too large, and (iii) there are not enough number of time steps, there does not exist such a trajectory that can cover the region and not exceed too much the boundary. To penalize the overpainted region, there would be some dark parts showing up in the result. However, if N is too large and the computation power is allowed, this wouldn't affect the smoothness or the visualization result. Therefore, we suggest that using relatively large N in each region if computation cost is allowed. It is also suggested to use different N in each sub-region dependent on the complexity of the shape of this sub-region and the total area of this sub-region.

- **Painting continuous colorful image.**
In the example, we show 9 examples of the painting continuous colorful image in Figure 10, 11,12,13, 14, 15, 16, 17, 18. The original images are the output of `CLIP` from inputting the sentence *A small girl on the grass play*. In each figure, the top left shows the normalized original image. The top right shows the segmented image from unsupervised segmentation task. In each sub-region, a trajectory of 8 time steps is learned for painting after 80 iterations. The final painted image is shown in the bottom left. The reflected trajectories are shown in bottom right. The learning rate for (x_i^j, y_i^j) and σ_i^j are 100 and 0.004. The parameter $\alpha = 10, \beta = 0.05$. A decent result can be obtained with 1 minute. It is shown that the

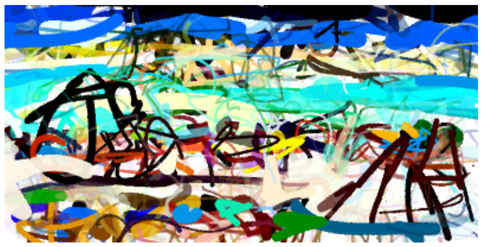


(a)

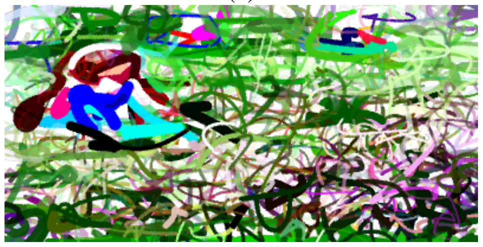


(b)

Figure 7. An example of painting colorful image with disconnected region with unified background. (a) Optimized image and target image. (b) Converging status of target image from initial after every 83 iterations. It is shown that after 249 iteration, the general shape of the object is found.



(a)



(b)

Figure 8. Simulated paintings generated with "Stage I and II combined" method using prompts (a) "A drawing of a beach with a pier on a cloudless day" and (b) "A small girl on the grass play"

proposed method can learn separated trajectories to roughly paint an image as a response to a sentence. It is also shown that the result is dependent on the segmentation task. Further work needs to be done to perform suitable segmentation tasks for robot painting.

3.3. Stage II -B

Figure 19 shows an example of using diffvg to produce the trajectories based on the image in generated raster image from Figure 10 (with the prompt "*A small girl on the grass play*"). Shown in Figure 19 are vector representations with varying numbers of strokes from 64 to 1024. Because the painting process is time consuming and has limited precision, we seek the fewest number of strokes that still produces a reasonable result. It is the belief of the authors that even 64 strokes produces good results.

The xy trajectories that the robot must follow are also shown in Figure 19 and are evidenced to be smooth and well-defined.

One limitation, that we attempted to address but were unable to obtain good results for, was to limit the total number of colors available in the "color palette". This is relevant to our application of graffiti painting since spray paint is difficult to mix so often there is a predefined set of 24 colors that are available. However, when allowing diffvg to produce general paintings, every stroke is almost guaranteed to have a different color. Constraining strokes to share colors (e.g. "weight sharing") or to have fixed colors from a palette are potential approaches. Assigning strokes colors from a discrete set of color options is an alternative approach that is likely to be very challenging due to the difficulty of hybrid optimization problems (those involving both discrete and continuous variables).

Another limitation is that many graffiti artworks are stylized by outline drawings, whereas computing vector drawings to match a photorealistic picture will not generate outline drawings. We hope our "Stage I and II combined" will fare better at these types of stylizations.

3.4. End-to-end Stage I and II combined

A selection of simulated images generated using our "Stage I and II combined" method are shown in Figure 8, and a comparison of results using different numbers of strokes can be found in Figure 20. Some additional results can also be found in the powerpoint file uploaded as part of the "code".

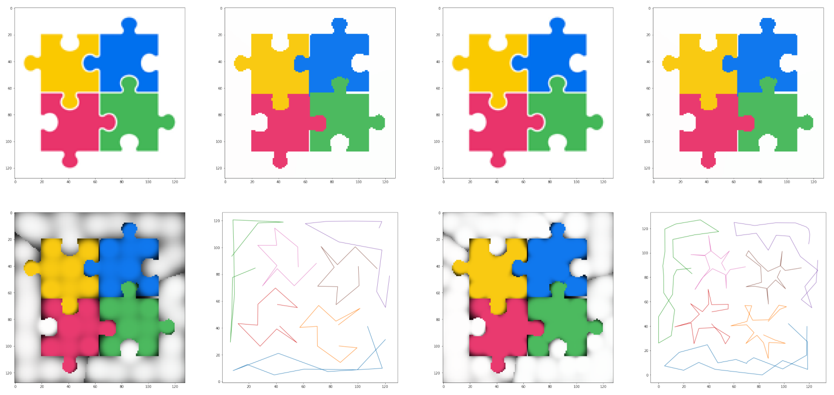

(a) $N = 10$ (b) $N = 20$

Figure 9. [Effect of the number of time steps N] An example of learning trajectory of discontinues colorful image including the background. In (a) and (b): Upper left is the original image. Upper right: Segmented image by using Pillow, where each segment(region) can be represented by using a unified color defined by the average of the color in this region in the original image. (8 segments). Bottom left: the output image painted by a robot. Bottom right: the corresponding trajectories designed for the robot to follow. In this example, we show that the proposed method can paint a decent close image with N sufficient large.

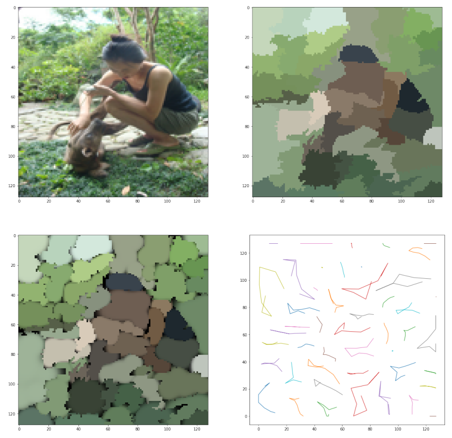


Figure 10. Example 1 - A small girl on the grass play. Upper left: output image from Clip by inputting the sentence *A small girl on the grass play*. Upper right: Segmented image by using Pillow, where each segment(region) can be represented by using a unified color defined by the average of the color in this region in the original image. (47 segments). Bottom left: the output image painted by a robot. Bottom right: the corresponding trajectories designed for the robot to follow.

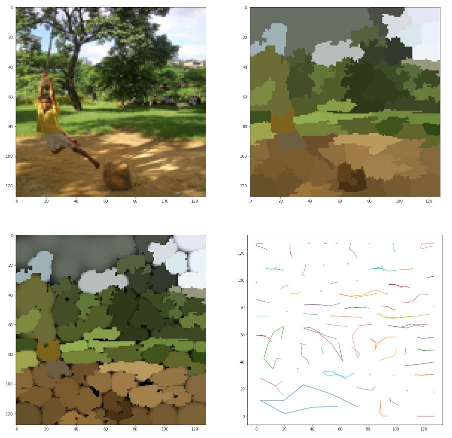


Figure 11. Example 2 - A small girl on the grass play. Output image from Clip, segmented image (62 segments), output "painting", and robot trajectories.

Generating good paintings is very difficult. A hypothesized large contributor to this is that, once a stroke color has been chosen and the stroke has grown sufficiently large, it becomes very difficult to change the color since the solution has fallen into a basin of attraction around that color. This manifests in many spurious strokes of random colors that should not be made, but cannot be optimized out because changing the color by differential amounts will not necessarily make the image semantically closer to matching the target sentence. This is made more obvious by watching videos of the optimization process, in which the colors of the strokes have been mostly decided after only a few dozen iterations. Another hypothesized contributor is the non-smoothness of the landscape - there is not an obvious smooth path from a random collection of lines to a semantically meaningful painting. In contrast, matching a raster image is much easier since me can get each of the strokes to better match the colors and shapes underneath it. Moving a squiggle by a tiny amount doesn't make it any more or less close to a target sentence.

Observing the optimization process in video form, it is clear that the background colors almost always converge very quickly (a less than a dozen iterations) and the few outlier colored strokes, if we get lucky, get optimized into foreground objects. For example, prompts that include words like "grass" or "beach" will cause almost all the strokes to converge very quick to green or blue/tan respectively. Typi-

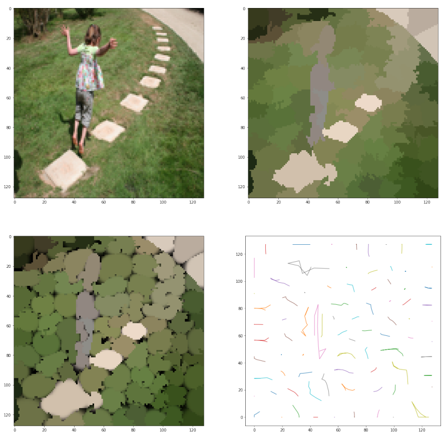


Figure 12. Example 3 - A small girl on the grass play. Output image from Clip, segmented image (81 segments), output “painting”, and robot trajectories.

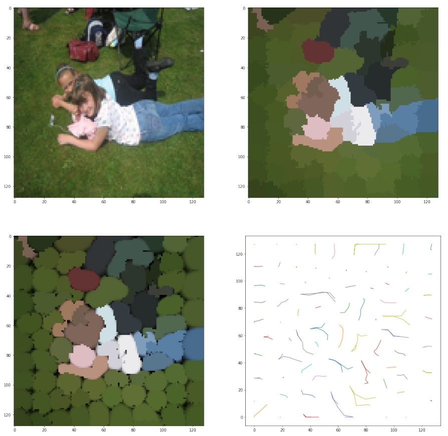


Figure 13. Example 4 - A small girl on the grass play. Output image from Clip, segmented image (75 segments), output “painting”, and robot trajectories.

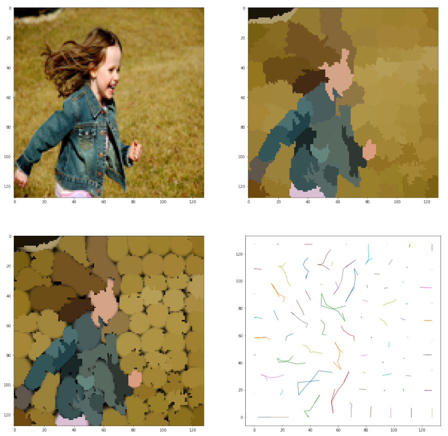


Figure 14. Example 5 - A small girl on the grass play. Output image from Clip, segmented image (77 segments), output “painting”, and robot trajectories.

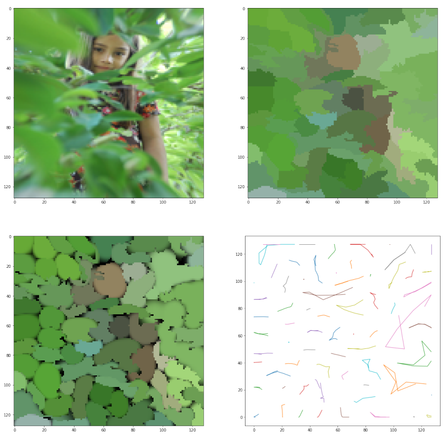


Figure 15. Example 6 - A small girl on the grass play. Output image from Clip, segmented image (68 segments), output “painting”, and robot trajectories.

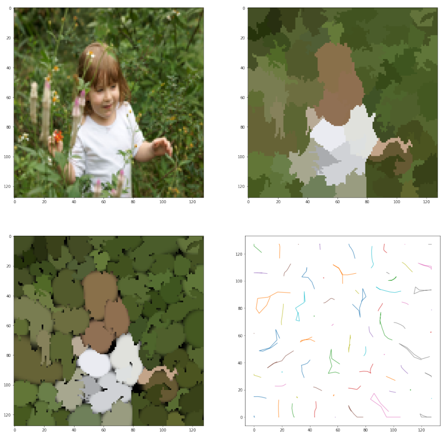


Figure 16. Example 7 - A small girl on the grass play. Output image from Clip, segmented image (68 segments), output “painting”, and robot trajectories.

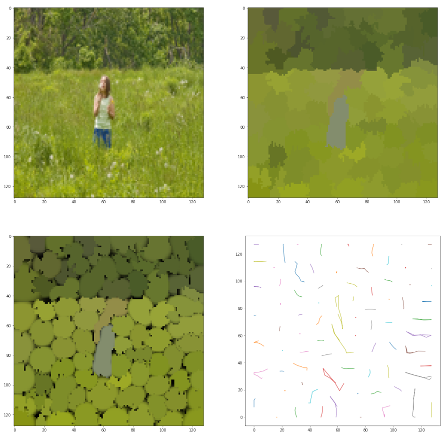


Figure 17. Example 8 - A small girl on the grass play. Output image from Clip, segmented image (68 segments), output “painting”, and robot trajectories.

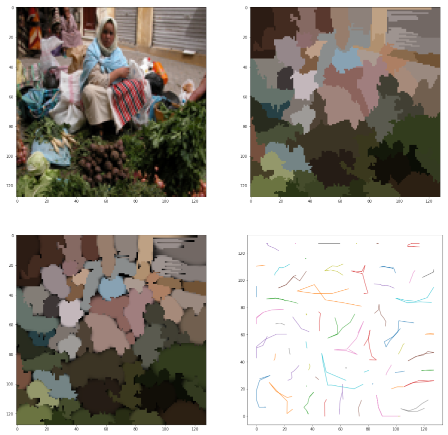


Figure 18. Example 9 - A small girl on the grass play. Output image from Clip, segmented image (59 segments), output “painting”, and robot trajectories.

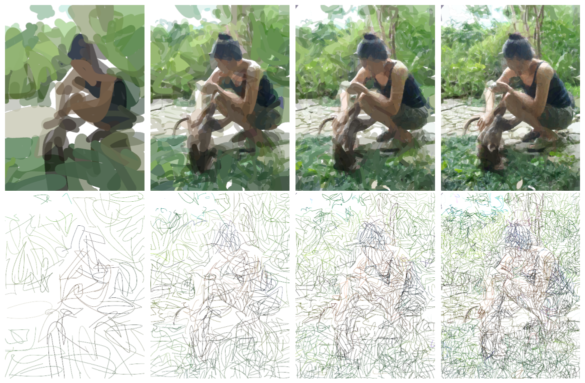


Figure 19. Rendered paintings of trajectories optimized by diffvg using (left-to-right) 64, 256, 512, and 1024 strokes. On the top row are the rendered paintings and on the bottom row are the robot trajectories with colors.

cally the large-scale structure image has more-or-less stabilized after around 100 iterations, with further iterations only causing strokes to slide around slightly (presumably due to the randomness inherent in the augmentations).

Some approaches that may improve performance include using a background color suitable to the setting, initializing stroke locations based on an existing image, and starting with many random initializations and selecting only a few to continue with after just a few dozen iterations (based on minimum loss), i.e. random-restarts.

4. Conclusion and Disscussion

In this report, we presented a novel methodology to produce a vectorized image from a given sentence. We implemented combing CLIP and Diffvq and showed experimental results based on this. We also proposed a simple method to replace Diffvq and produce a set of trajectories which can be represented by

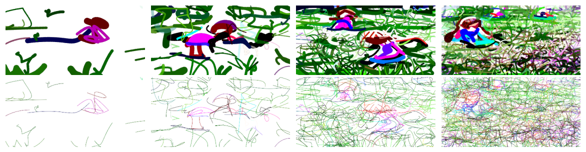


Figure 20. Rendered paintings of trajectories optimized by diffvg+CLIP for the prompt “A small girl on the grass play” using (left-to-right) 16, 64, 256, and 1024 strokes. On the top row are the rendered paintings and on the bottom row are the robot trajectories with colors.

1. Since we con considering learning the trajectories in each region Ω_k, We suggest considering parallel Computing to perform multiple tasks at the same time.

2. Based on the fact that the painting result would be highly dependent on the segmentation result, we suggest exploring better segmentation or image vectorization method.

3. The proposed scheme can be modified such that not only closed curves are preferred.

4. Regarding the loss function. We explore the curvature term. However, due to the boundary condition in Equation (7), the curvature cannot be defined in some cases. However, this is still useful if a robot prefers to turn smoothly than sharply.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[2] G. Chen, S. Baek, J.-D. Florez, W. Qian, S. won Leigh, S. Hutchinson, and F. Dellaert. Extended version of GTGraffiti: Spray painting graffiti art from human painting motions with a cable driven parallel robot, 2021.

[3] K.-W. Chen, Y.-S. Luo, Y.-C. Lai, Y.-L. Chen, C.-Y. Yao, H. kuo Chu, and T.-Y. Lee. Image vectorization with real-time thin-plate spline. *IEEE Transactions on Multimedia*, 22:15–29, 2020.

[4] K. Crowson. Vqgan+clip(updated). `https://colab.research.google.com/github/justinjohn0306/VQGAN-CLIP/blob/main/VQGAN%2BCLIP%28Updated%29.ipynb?authuser=3`.

[5] K. Frans, L. B. Soros, and O. Witkowski. Clipdraw: Exploring text-to-drawing synthesis through language-image encoders, 2021.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] F. Hou, Q. Sun, Z. Fang, Y.-J. Liu, S.-M. Hu, H. Qin, A. Hao, and Y. He. Poisson vector graphics (pvg). *IEEE transactions on visualization and computer graphics*, 26(2):1361–1371, 2018.

[8] B. Irving. maskslic: regional superpixel generation with application to local pathology characterisation in medical images. *arXiv preprint arXiv:1606.09518*, 2016.

[9] T.-M. Li, M. Lukáč, G. Michaël, and J. Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 39(6):193:1–193:15, 2020.

[10] W. Li, P. Zhang, L. Zhang, Q. Huang, X. He, S. Lyu, and J. Gao. Object-driven text-to-image synthesis via adversarial training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12174–12182, 2019.

[11] Y. Li, X. Zhai, F. Hou, Y. Liu, A. Hao, and H. Qin. Vectorized painting with temporal diffusion curves. *IEEE transactions on visualization and computer graphics*, 27(1):228–240, 2019.

[12] S. Lu, W. Jiang, X. Ding, C. S. Kaplan, X. Jin, F. Gao, and J. Chen. Depth-aware image vectorization and editing. *The Visual Computer*, 35(6):1027–1039, 2019.

[13] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021.

[14] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[15] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *International Conference on Machine Learning*, pages 1060–1069. PMLR, 2016.

[16] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[17] M. Tao, H. Tang, S. Wu, N. Sebe, X.-Y. Jing, F. Wu, and B. Bao. Df-gan: Deep fusion generative adversarial networks for text-to-image synthesis. *arXiv preprint arXiv:2008.05865*, 2020.

[18] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915, 2017.

[19] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1947–1962, 2018.

[20] M. Zhu, P. Pan, W. Chen, and Y. Yang. Dm-gan: Dynamic memory generative adversarial networks for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5802–5810, 2019.