Laplacian-steered stylized neural painting

Ayush Agarwal*, Akshat Goyal †, Andreas Rosenmeier ‡, Sankalan Baidya§ and Soham Joshi¶
Department of Computer Science, IIT Bombay
Mumbai, Maharashtra, India
Email: * ayushagarwal@cse.iitb.ac.in, † akshatgoyal@cse.iitb.ac.in, ‡ 23v0018@iitb.ac.in, § somekoln@cse.iitb.ac.in, ¶ sohamjoshi@cse.iitb.ac.in

***Abstract*—This work builds upon the foundations of Stylized Neural Painting [7] by introducing a lightweight model with fewer parameters and optimizing code through vectorization, aiming to expedite the process of style transfer. Additional experiments were conducted by augmenting the code with a novel term in the loss function, inspired by the Laplacian loss proposed in [5]. Further we have extended our model for style transfer on videos.**

***Index Terms*—Neural Networks, Deep Learning, Image Processing, Style Transfer, Convolutional Neural Networks (CNN), Feature Extraction, Artistic Style, Image Synthesis**

I. Introduction

Artistic capabilities have been traditionally viewed as belonging exclusively to humans. However, recent progress has shown that this is not the case. Previously, progress has been made in generating art using a pixel-by-pixel approach [3] [6], or a continuous optimization process. However, neither of these seem to capture the art of brushstrokes seen in popular oil paintings. A breakthrough paper in this respect was published recently [7] which showed an automatic image-to-painting translation method that generates vivid and realistic paintings with controllable styles. This method was referred to as "Stylized Neural Painter". Instead of manipulating pixels, this method generated brush strokes just like a human painter would

Recently, there was also reported success [5] in using the Laplacian loss in neural style transfer which steers the synthesized image towards having similar low-level structures as the content image, while being flexible to allow the image to be rendered in the new style. The third frontier we shall focus on is neural style transfer for videos. Recently progress has been made on this front by [2].

Our work aims to combine work from these frontiers. In order to achieve this, we introduce the laplacian loss function in the framework proposed by [7], in order to improve the peak signal to noise ratio (PSNR) for neural style transfer on the stylized neural painted images. Moreover, we run the Stylized neural painter frame by frame in order to generated transformed videos. We observe that in practice that this simplistic approach works when working on videos lasting a few seconds. Further, we propose a more succint model, which renders images faster than the original model.

II. Preliminaries

In this report, we mainly build upon the works of [5] and [7]. So, in order to understand our code, we need some mathematics background to motivate the changes we made in the code of [5]. In this section, we enlist the mathematical framework of the above mentioned papers.

A. Stylized Neural Painting

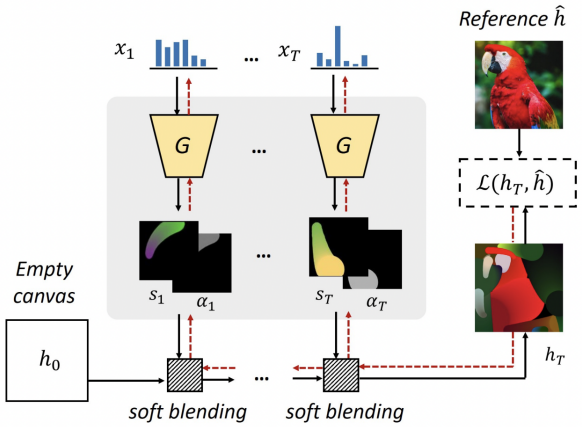


Fig. 1: Network architecture taken from [7]. We start from an empty canvas and then render stroke-by-stroke with soft blending. We use gradient descent to find a set of "optimal" stroke parameters that minimize the loss. Here black arrow lines mean forward propagation and red ones mean back-propagation of the gradient

The overview of the method used by [7] is depicted in fig. 1. In each drawing step t, a trained neural renderer G takes in a set of stroke parameters x_t (e.g., shape, color, transparency, and texture), and produces a stroke

foreground s_t and an alpha matte α_t. We then use soft blending to mix the canvas, the foreground, and alpha matte at each step t and make sure the entire rendering pipeline is differentiable. The soft blending is defined as follows,

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

where $(s_t, \alpha_t) = G(x_t)$. The stroke parameters are now gathered from all the T steps and optimize them by searching within the stroke parameter space. The searching is conducted under a self-supervised manner, i.e., it is enforced that the final rendered output h_T similar to a reference image $\hat{h}$:

$$h_T = f_{t=1,T}(\bar{x})$$

where $f_{t=1,T}(.)$ is a recursive mapping from stroke parameters to the rendered canvas. $\bar{x} = [x_1, \cdots, x_T]$ are the collection of stroke parameters at $t = 1, \cdots, T$ drawing steps

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

$$\bar{x} \leftarrow \bar{x} - \mu \frac{\partial \mathcal{L}(h_T, \hat{h})}{\partial \bar{x}}$$

B. Disentangle neural rendering

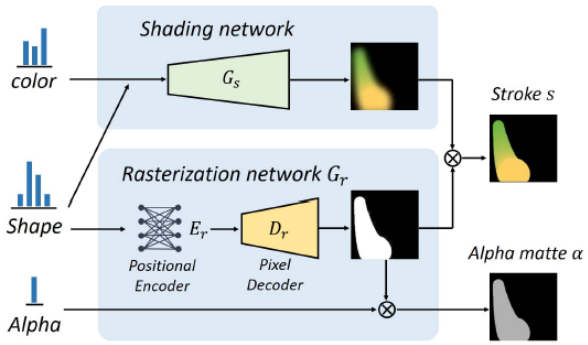


Fig. 2: [7] designs a dual-pathway neural renderer which consists of a shading network Gs and a rasterization network Gr. This renderer takes in a group of stroke parameters (color, shape, and transparency) and produces the rasterized foreground map and alpha matte.

Previous renderers suffered from coupling of shape and color representations when testing with more complex rendering settings like color transitions and stroke textures. This problem was solved by zou et. al. using the architecture depicted in Fig. 2 which employs a dual pathway neural renderer that disentangles color and shape through the rendering pipeline. This is achieved using a shading network G_s and a rasterisation network G_r. The parameters of a stroke x are divided into three groups: color, shape, and transparency. G_s is built as a stack of several transposed convolution layers, which takes in both the color and shape parameters and generates strokes with faithful foreground color. They design the G_r as a positional encoder + a pixel decoder, which simply ignores the color but generates stroke silhouette with a clear shape boundary. We finally generate the output stroke foregrounds by masking the color map with the stroke silhouette and generate the final alpha matte α by rescaling the silhouette using the input alpha value. This neural renderer is trained with standard l_2 pixel regression losses on both the rendered stroke foreground and the alpha matte. During the training, the following objective function is minimized :

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

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

C. Laplacian-Steered Neural Style Transfer

The Laplacian operator Δ of a function f is the sum of all unmixed second partial derivatives:

$$\Delta f = \sum_i \frac{\partial^2 f}{\partial x_i^2}$$

The Laplacian filter is the discrete approximation to the two dimensional Laplacian operator, given by

$$D = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

The Laplacian matrix of an image x is obtained by convolving the image with D, denoted by D(x). At regions where adjacent pixel values change drastically, the convolution will produce a response of high magnitude, regardless of the direction of the change. In regions where the change is flat, the response is zero. Hence the Laplacian operator is widely used for edge detection and extraction of detail structures in an image

1) Laplacian Loss and objecive: Given two images x_c and $x^{'}$ we can use a Laplacian loss $\mathcal{L}_{lap}$ to measure the difference between their Laplacian

$$\mathcal{L}_{lap} = \sum_{ij} \Big(D(x_c) - D(x^{'}) \Big)_{ij}^2$$

In order to force the stylized image to possess similar detail structures as the content image, we can augment the style transfer objective with the Laplacian loss $\mathcal{L}_{lap}$ of $x^{'}$ and x_c :

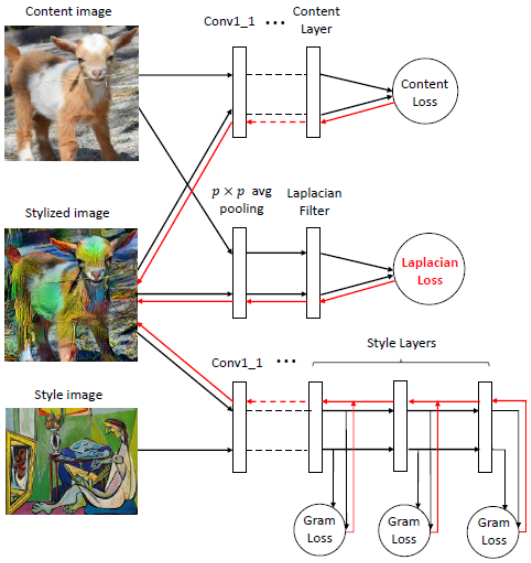


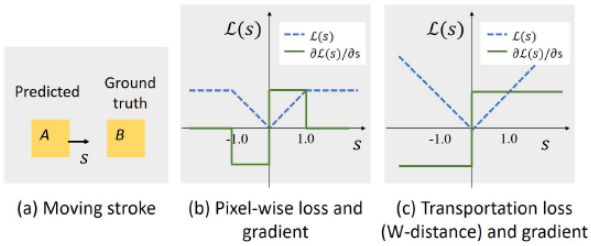
Fig. 3: Network Architecture of Lapstyle [5]. Black and red lines are forward and backward passes, respectively. Dashed lines indicate that there are unshown intermediate layers.

$$\mathcal{L}_{total} = \alpha \cdot \mathcal{L}_{content} + \beta \cdot \mathcal{L}_{style} + \gamma \cdot \mathcal{L}_{lap}$$

In order to implement this, the neural network *Lapstyle* given in [5] uses the above linear combination given as 3

D. Pixel Loss and zero gradients

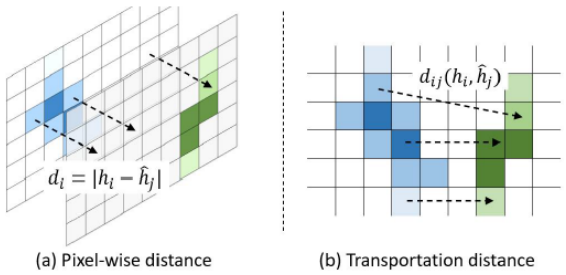
This section describes the zero gradient problem, that is, why l_1 loss is not sufficient to capture our problem. The below figures are taken from [7].



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

Fig. 4: A simple example to explain why pixel-wise loss L may have the zero-gradient problem ($\partial L/\partial s = 0$) during the stroke update.

Suppose we have a square shaped stroke A and we want it to converge to a target location B. Now, the stoke parameters allow you to shift the stroke horizontally. Now, since there is no overlap between the target and the source strokes, the gradient corresponding to that parameter becomes zero. This phenomenon has been depicted in fig. 6. In order to rectify this issue, we need to define a transportation metric instead of a pixel-wise distance metric as shown in fig. 5.



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

Fig. 5: A comparison between the pixel-wise distance and the transportation distance

If instead, the loss is defined as the amount of transportation effort, as shown in fig. 4, we see that the issue is fixed.

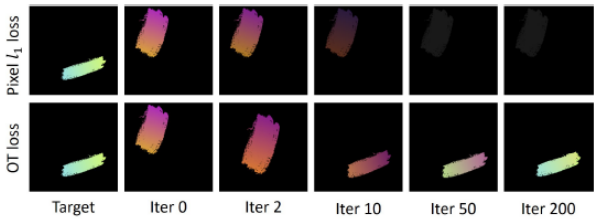


Fig. 6: A comparison between the pixel loss (1st row) and transportation loss (2nd row) on "pushing" a stroke from its initial state to its target. Using the proposed transportation loss, the stroke nicely converged to the target. As a comparison, the pixel loss fails to converge due to the zero-gradient problem in its position and scale.

We define the minimum transportation efforts, i.e., the Wasserstein distance, as an effective measure of similarity loss between the canvas and the reference image.

Given a rendered canvas h and a reference image $\hat{h}$, we define their normalized pixel values p_h and $\hat{p_h}$ as their probabilistic marginal functions. Here we omit the subscript T for simplicity. We define $P \in \mathbb{R}_{+}^{n \times n}$ as the joint probability matrix whose (i, j)-th element denotes the joint probability of the i-th pixel in h and j-th pixel in $\hat{h}$, where n is the number of pixels in the image. We let D be the cost matrix whose (i, j)-th element denotes the Euclidean distance between the i-th pixel's location in h

and j-th pixel's location in $\hat{h}$. Thus, the matrix D list all the labor costs of moving a "unit pixel mass" from one position in h to another one in $\hat{h}$. In the discrete case, the classic optimal transport distance can be written as a linear optimization problem

$$\min_{P \in \mathcal{U}} D^T P$$

where $\mathcal{U} := \{P \in \mathbb{R}_+^{n \times n} \text{ s.t. } P1_n = p_h, P^T 1_n = \hat{p_h}\}$ In [7], a smoothed version of the above constrained optimization problem called the sinkhorn distance is employed which has several nice properties including differentiability, and an entropic regularization term. Let $\mathcal{L}_{ot}$ be the sinkhorn distance defined as the following.

$$\mathcal{L}_{ot}(h, \hat{h}) := D^T \tilde{P_\lambda}$$
$$\tilde{P_\lambda} = \arg\min_{P \in \mathcal{U}} D^T P - \frac{1}{\lambda} E(P)$$

where $E(P) := -\sum_{i,j=1}^{n} P_{i,j} \log P_{i,j}$, and the optimized transport loss can be defined as

$$\mathcal{L} = \beta_1 \mathcal{L}_{l_1} + \beta_2 \mathcal{L}_{ot}$$

III. Modifications

A. *Laplacian*

In recent work by Li [5], the idea of laplacian steered neural style transfer was explored. In order to improve the structural coherence, the paper argues that the content loss reduces the search space too much, whereas the laplacian is quite flexible and can be added to the loss without disrupting the existing structure. We test the same theory with this modification, where we assign the laplacian loss term a fixed weight, so that the loss function now becomes,

$$\mathcal{L}_{total} = L_{content} + \alpha \mathcal{L}_{l_1} + \beta \mathcal{L}_{style} + \gamma \mathcal{L}_{lap} + \delta \mathcal{L}_{ot}$$

that is a linear combination of content, style, laplacian and sinkhorn loss functions, where the coefficient of the laplacian loss is fixed. This laplacian loss is simply a sum of the laplacian losses with respect to a selection of layers corresponding to the neural network generating the deep representation of the image (*vgg-16*) in our case. Note that we are not using laplacian loss in the rendering of the image, but in order to apply a style to the rendered image.

B. *Lightweight Renderer*

Drawing from the renderer given in the paper, we designed our own renderer with both the shader and the rasterizer network. We greatly reduced the number of parameters from 18.1 million to 5.4 million. Our architecture for the networks is shown below. In the config column entries of the form $c \times w \times w/s$ represents c filters of size $w \times w$ being applied with stride s, the output is $height \times width$.

Table 1: Details of our shading network.

	Layer	Config	Out size
C1	Deconv + BN + ReLU	$512 \times 4 \times 4/1$	$4 \times 4 \times 512$
C2	Deconv + BN + ReLU	$512 \times 4 \times 4/4$	$16 \times 16 \times 256$
C3	Deconv + BN + ReLU	$256 \times 4 \times 4/4$	$64 \times 64 \times 256$
C4	Deconv + BN + ReLU	$128 \times 4 \times 4/2$	$128 \times 128 \times 6$

Table 2: Details of our rasterization network.

	Layer	Config	Out size
F1	Full-connected + ReLU	512	512
F4	Full-connected + ReLU	4096	4096
V1	View	–	$16 \times 16 \times 16$
C1	Conv + ReLU	$32 \times 3 \times 3/1$	$16 \times 16 \times 32$
C2	Conv + Shuffle	$32 \times 3 \times 3/2$	$32 \times 32 \times 8$
C3	Conv + ReLU	$16 \times 3 \times 3/1$	$32 \times 32 \times 16$
C4	Conv + Shuffle	$16 \times 3 \times 3/2$	$64 \times 64 \times 4$
C5	Conv + ReLU	$8 \times 3 \times 3/1$	$64 \times 64 \times 8$
C6	Conv + Shuffle	$4 \times 3 \times 3/2$	$128 \times 128 \times 1$

Note that in contrast, to the network given in [7] our network removes two fully connected layers F2 and F3 (according to [7] in the rasterizer) and removes C5 and C6 layers in the shader and modify C1 to C4. However, our result of the PSNR and G_loss (definitions of PSNR and G_loss elaborated in IV) show that our lightweight network with performs as good as their network.

C. *Stylized neural painting for videos*

The idea of neural style transfer for videos has been explored in [2], and other such papers. So it is natural to expect such an extension for this new kind of style transfer, where an image can be converted into its painted counterpart. In order to implement this, we perform the following procedure,

1) Given in input video, we use the *opencv* library to break the video into frames, and consider every other frame for the computation of the transformed frames to be efficient. So, the resultant video has half the initial frames, that is *fps* (frames per second) drops by half.
2) For each frame, we run the painting network, which gives us the transformed image
3) Finally, all the transformed frames are joined back together using the *opencv* library in python.

The video we got using this approach was not smooth, i.e. there was difference(mostly in texture) in two consecutive frames. This difference was due to the randomness

in the initial parameters and that many optimal stroke parameters are possible which give similar results. We tried adding L2 loss between a current frame and it's previous frame in our loss function for videos, however the results did not improve significantly, maybe owning to the fact that the frames are much alike pixelwise so the value of the added L2 loss is very less, the difference between consecutive strokes is felt in the texture, i.e. along the edges of strokes.

D. Code Optimization

We have built upon the code from the official implementation of [7]. In addition to the above major changes, some changes changes to specific sections of code were also made to make the code faster, work with better accuracy.

1) The `for` loops present in files `morphology.py` were removed and replaced with vectorised code, yielding a speedup of 7% on the original code
2) We attempted to use the network *vgg-19* instead of *vgg-16* in order to obtain deep representations of the images

IV. Experiments

Now that we have our modified set-up, we can describe the analysis we perform to assess the modified model. Moreover, since the model is meant for producing aesthetically pleasing images, which mimic human painting, we will observe many examples of rendered images.

A. Metrics

Before performing any experiments, we need to define metrics for how good the brush stroke representation of an image is. To this end, we mainly analyse the metric *PSNR* (peak signal to noise ratio). PSNR is most easily observed as a function of the *MSE* (mean squared error), where

$$\text{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

Now, PSNR is given by,

$$\begin{aligned}\text{PSNR} &= 10\log_{10}(\frac{\text{MAX}_I^2}{\text{MSE}}) \\ &= 20\log_{10}(\text{MAX}_I) - 10\log_{10}(\text{MSE})\end{aligned}$$

where, MAX_I is the maximum possible pixel value of the image. So, PSNR is morally just the mean squared error.

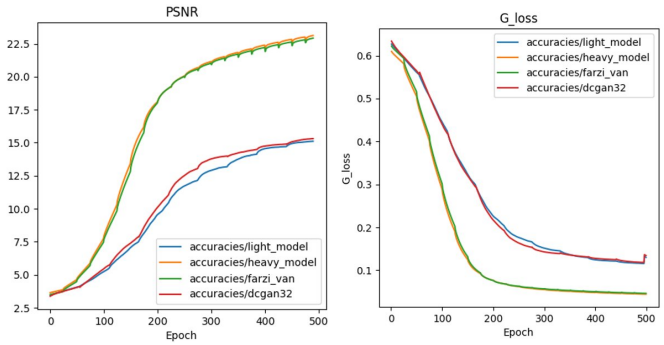


Fig. 7: Comparison between PSNR and G_loss values for different renderer architectures,(a) Light model provided in zou et. al, (b) Heavy model provided in zou. et. al for oil painting, (c) Our modified model & (d) DCGAN-32

B. Evaluating the modified rendering network

In order to compare performances, the performance metric PSNR. The comparison of some of the transformed images is shown in the table

In order to compare the PSNR, we followed the procedure given in [7], and we considered the mean PSNR obtained over all the images in the dataset and plotted it against the number of epochs. Since PSNR is a metric of similarity between images, higher PSNR implies a better model. In the analysis here, we consider four networks for the plotting of PSNR, loss curves 7 namely, the provided light, heavy models in [7] for oil painting, dcgan32 and our model.

We can infer from the PSNR curves that the PSNR for our model is approximately same as the heavy model, and this fact is evident from the rendered images seen before. Moreover, our model has three times less parameters than the original model, making this model all the more efficient.

C. Evaluating the effect of Laplacian Loss

Now that we have analysed the rendering aspect of our modifications, we dive into style transfer. That is, given a style image and a rendered (painted) image, the task boils down to transferring this style to the rendered image. In order to do this, we have modified the loss function to include a laplacian term as described previously. This analysis is carried out by comparing PSNR values for the style transferred image. Low values of PSNR are good for this application since the modified image needs to have a component of the style, and hence needs to be "far" from the original image.

Hence, we analyse two variations of our model for style transfer, one with laplacian and one without. Upon

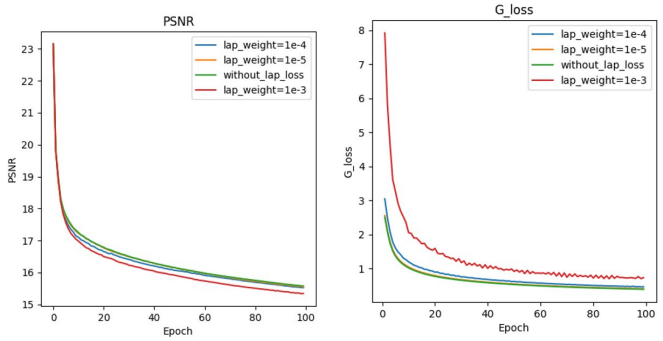


Fig. 8: Comparison between PSNR and G_loss values for different weights of the Laplacian loss

observation of psnr for these variations we get the plots shown in fig. 8.

This shows that models with laplacian losses are better since a higher weight of laplacian loss reduces the PSNR marginally. Upon plotting the loss curves as shown in fig. 8 we get that higher weight of laplacian is correlated with a higher value of loss function, which is expected since we are adding an extra term in the loss and the metric to be actually compared is the PSNR.

D. Speed-ups observed

Upon slight modifications to the code as described earlier, and running the code several times on google colab with T4 GPU runtime, and taking the mean, we observe the following speedups :

1) The time for rendering an image in our model, when compared with the original heavy model for oil-painting, gives a speed-up of approximately 5% (here, the flags `save_jpg` and `save_video` have been set to False)
2) In our model, upon vectorising the for loops given in `morphology.py`, we observe a speed-up of 7%

Note that the running time of our model is still significantly slower than the light model provided in [7], so there is a trade-off between psnr and running time, number of learnable parameters. Now, moving on to discussion of the results we have obtained.

V. Discussion

We build upon the progress on the nature of human painting using differentiable stroke rendering. The method can generate realistic artworks for most reference images. Moreover, we examine style transfer on these rendered images, exploring the effects of a laplacian loss term. We further extend this by performing style transfer of videos. We achieve some amount of code optimization by reducing the running time of the original code by 7%. Moreover, we propose a modified model, which takes three times less parameters than the original model and runs with a 5% speedup.

Some directions of future progress include adding a precomputed flow function in order to get smoother videos. Further, the idea of neural style transfer in its genesis [1], crucially used the fact that a style loss term is added in the loss function, which is the grammian of a deep feature representation. But why do we restrict ourselves to only second order moments in the form of quadratic kernels? This question has been examined in [4], which showed style transfer using higher order moments.

VI. Acknowledgements

We thank Prof. Preethi Jyothi and her teaching assistants for the opportunity to undertake this project as a part of the course *Artifical Intelligence and Machine Learning* offered in Autumn '23 at Indian Institute of Technology, Bombay.

References

[1] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016.

[2] Haozhi Huang, Hao Wang, Wenhan Luo, Lin Ma, Wenhao Jiang, Xiaolong Zhu, Zhifeng Li, and Wei Liu. Real-time neural style transfer for videos. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7044–7052, 2017.

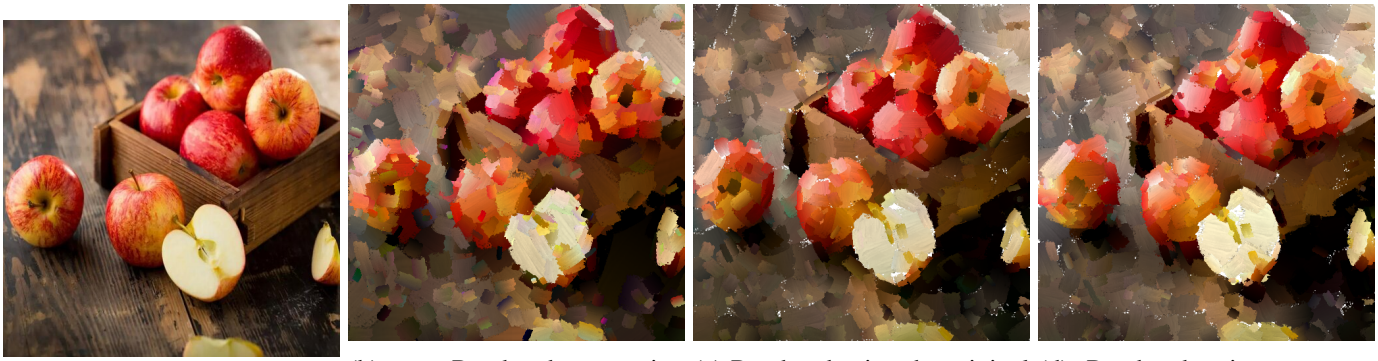
[3] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[4] Nikolai Kalischek, Jan Dirk Wegner, and Konrad Schindler. In the light of feature distributions: moment matching for neural style transfer, 2021.

[5] Shaohua Li, Xinxing Xu, Liqiang Nie, and Tat-Seng Chua. Laplacian-steered neural style transfer. In *Proceedings of the 25th ACM international conference on Multimedia*, MM '17. ACM, October 2017.

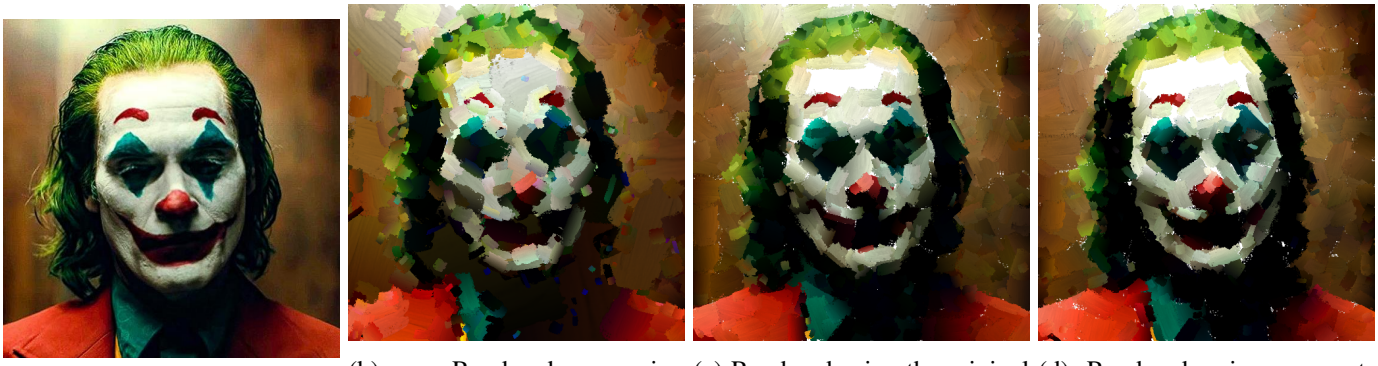
[6] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.

[7] Zhengxia Zou, Tianyang Shi, Shuang Qiu, Yi Yuan, and Zhenwei Shi. Stylized neural painting, 2020.



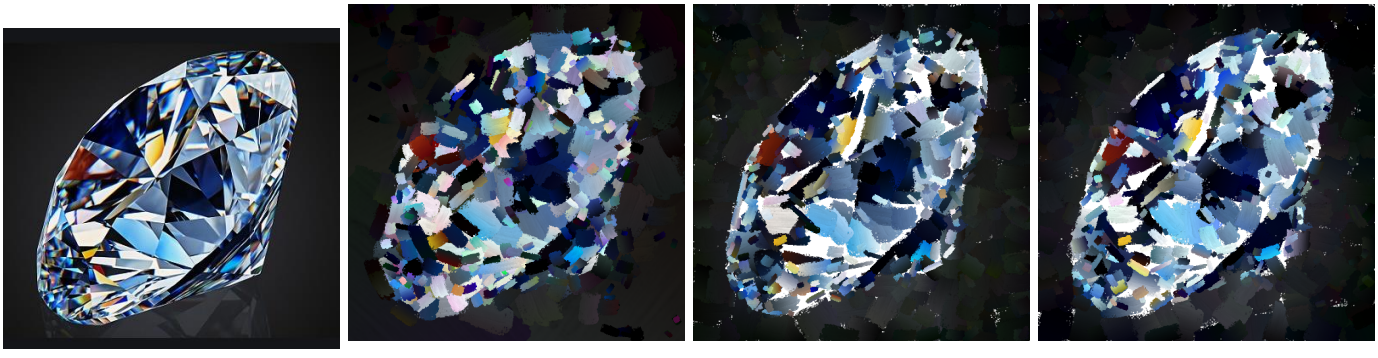
(a) Original Image (b) Rendered using DCGAN_32 (c) Rendered using the original network (d) Rendered using our network

Fig. 9: Showing Rendering using different network architectures



(a) Original Image (b) Rendered using DCGAN_32 (c) Rendered using the original network (d) Rendered using our network

Fig. 10: Showing Rendering using different network architectures

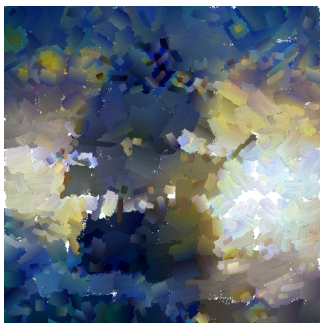


(a) Original Image (b) Rendered using DCGAN_32 (c) Rendered using the original network (d) Rendered using our network

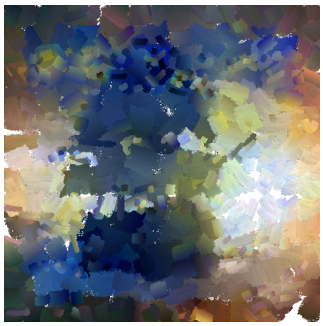
Fig. 11: Showing Rendering using different network architectures

(a) Original Image

(b) Style Image



(c) Original Renderer

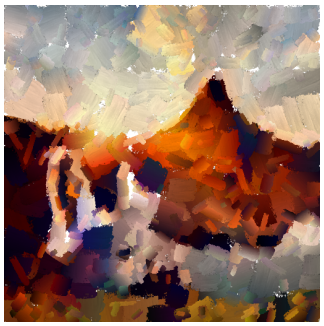


(d) Our Renderer with Laplacian

Fig. 12: Showing Style transfer using our network and the original network

(a) Original Image

(b) Style Image



(c) Original Renderer

(d) Our Renderer with Laplacian

Fig. 13: Showing Style transfer using our network and the original network

(a) Original Image

(b) Style Image

(c) Original Renderer

(d) Our Renderer with Laplacian

Fig. 14: Showing Style transfer using our network and the original network