INSTITUTE OF INFORMATION TECHNOLOGY

DeepPaint

An AI powered tool for Digital Painting

Shihab Shahriar Khan

13-Dec-18

DeepPaint

Submitted To

Dr. Naushin Nower
Assistant Professor

Institute of Information technology
University of Dhaka

Supervised By

Dr. Ahmedul Kabir
Assistant Professor

Institute of Information technology
University of Dhaka

Letter of Transmittal

13th December, 2018
Coordinator
Software Project Lab 3
Institute of Information Technology
University of Dhaka

Subject: Submission of technical report on “DeepPaint: AI powered digital painting assistant”

Sir,
I, Shihab Shahriar Khan, am submitting my technical report with due respect. Despite my best effort, it might still contain some errors. I hope that you would be kind enough to accept this report.

Yours sincerely
Shihab Shahriar Khan
BSSE 0703
Institute of Information Technology
University of Dhaka

Abstract

This project provides users two powerful tools to aid in digital painting creation: namely colorizing and stylizing. First tool allows user to delegate the task of coloring a black and white painting to a deep learning-based AI agent, which is capable of understanding image content and color them using knowledge gathered from thousands of images. Second tool helps to stylize the painting, by subtly manipulating the painting along the famous artistic styles like Expressionism, Cubism or the style of painters like Van Gogh or Claude Monet. Together they tackle two of the most challenging tasks in digital painting, both from the perspective of computers and human artists.

Acknowledgment

I want to start by thanking my supervisor, Ahmedul Kabir, for constant guidance and helping me overcome the obstacles of this project.

I want to thank director of IIT for allowing me to use computing resources from Master's lab, and seniors of MSSE for accommodating me.

And I thank IIT for giving us the freedom, guidance and resources to work on such interesting projects, and for this overall illuminating experience.

Letter of Endorsement

The SPL3 project report titled, DeepPaint, has been submitted to me prior to the final submission. I have gone through all the contents of the report and found that to the best of my knowledge, the information presented here is factually accurate. I hereby gladly assert the validity of this report and wish a bright future for Shihab Shahriar Khan.

Dr. Ahmedul Kabir,

Assistant Professor,

Institute of Information Technology,

University of Dhaka

Contents

Table of figures

Deep Paint

1 Introduction

DeepPaint is an artificial intelligence powered computer vision tool that hopes to help amateur artists overcome their lack of artistic skills and bring their imagination to reality.

This web tool helps users to automate every aspect of colorizing and stylizing a painting, while simultaneously giving user some control to customize these processes in her own way. It leverages the recent advances in computer vision, mainly centered on deep learning. It is simple and designed to be very easy to use, so newcomers don't have to struggle through a steep learning curve to get started.

DeepPaint differentiates itself from more traditional digital editing softwares by its highly advanced ability to understand semantic content of images and manipulating them in meaningful way. Also, it focuses on few important services, instead of providing a comprehensive set of tools needed by digital artists. Overall, it aims to complement traditional softwares, not replace them.

2 Product Overview

2.1 Component Overview

DeepPaint currently provides two functions: colorizing and stylizing an image. There is also a Gallery feature to help users organize their paintings.

2.1.1 Painting Colorizer

This component colorizes a grey scale image of painting. This can be done completely automatically. Since there isn't any single correct way to colorize an image, user can provide "hints" by colorizing few points on the image to guide the colorizer. Number of such hints can vary. However, for users with a particular colorization in mind more points usually help the colorizer to get closer to that imagined version.

2.1.2 Painting Stylizer

A style of an artwork refers to its distinctive visual elements, techniques and methods [1]. This component allows users to stylize their painting along particular popular styles like Impressionism,

Cubism and Gothic etc. Users can also stylize their painting by referring to the style of another painting.

2.1.3 Gallery

This component helps user to organize their paintings during different stages of editing. This acts as data source for other components, and as a bridge that controls communication among components. Its common functionality includes uploading/downloading paintings, deleting, selecting for modification etc.

2.2 USAGE SCENARIO

2.2.1 Gallery

User first uploads their painting in gallery. To carry out an operation on painting(s), users first have to select it from gallery (or drag it to that operation's component window). After operation is completed, the output can also be pushed to the gallery, maybe for further processing. Gallery allows limited number of paintings. Paintings can be deleted from gallery, and downloaded.

2.2.2 Painting Colorizer

User selects a greyscale image from gallery (any colored image will automatically be treated as greyscale). By using a color palette, she can place arbitrary number of hints i.e. colored points, on that image. Then a simple pressing of a button would produce the colored painting in a different window.

2.2.3 Painting Stylizer

User selects a painting from gallery to "content" box, and then a style from a set of predefined styles to "style" box. A pressing of "apply" button would then produce the stylized image. For stylization by referring to a particular image, that image first needs to be uploaded to gallery, and selected for the "style" box.

2.3 QFD

This section summarizes the product requirements.

2.3.1 Normal

- Automatic colorization of paintings.
- User guided colorization.
- Stylization using abstract styles.
- Stylization using a particular painting.
- Ability to work with multiple images concurrently.
- Ability to work with images of arbitrary size.

2.3.2 Expected

- Upload/Download painting from gallery.
- Send output painting to gallery.
- Delete/Sort content of gallery.

2.3.3 Exciting

- Colorize partially colored images.
- Add “drag & drop” feature to select painting from gallery.
- Real time visualization of multi-pass stylization.
- Real time visualization of colorization after every hint.
- Monitor resource usage.
- Change hint radius for colorization.

3 METHODOLOGY

3.1 USER GUIDED IMAGE COLORIZATION

A deep neural network is trained to convert any user provided greyscale image, along with sparse user hints, into colored one. The network automatically learns to propagate colored hints across entire image through a training process involving millions of images. Here we briefly describe the algorithm used [2].

Figure 1: Example of guided image colorization. Note the little color hints on greyscale image. (Image taken and edited from original paper [2])

3.1.1 Training

Input images are converted to CIE Lab color space. Network is trained in supervised fashion, with Lightness channel (L) extracted from *Lab* being the input to the network, and the output is color channels *ab*. A convolutional neural network F, parameterized by θ is used to learn this mapping.

The output colors $\hat{Y}$ is compared against ground truth Y to compute loss and readjust network parameters using variant of L1 loss function.

Goal of this training phase is to find optimal weight θ so that:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{X,U,Y\sim\mathcal{D}}[\mathcal{L}(\mathcal{F}(X,U;\theta),Y)]$$

Here L is the loss function, that compare F's output with ground truth Y, and D is the distribution of our dataset. Simply put, during training we try to find weights for our network that minimizes above loss.

The loss function in particular is a variant of L1 loss:

$$\ell_\delta(x,y) = \tfrac{1}{2}(x-y)^2 \mathbb{1}_{\{|x-y|<\delta\}} + \delta(|x-y| - \tfrac{1}{2}\delta)\mathbb{1}_{\{|x-y|\geq\delta\}}$$

Where 'x, y' comes from ground truth image and predicted image respectively.

To incorporate user hints, during training a random subset of points are concatenated with greyscale image to simulate sparse color hints. This size can be zero, to teach network completely automatic colorization. This subset can also equal the size of the whole image points, to help network learn identity transformation. Images used during training came from ImageNet [3] dataset.

3.1.2 Network Architecture

The neural network consists of 10 convolution blocks, each block have 2/3 pairs of convolution layer and ReLu activation layer. Each of the first 4 convolutional blocks compresses image by halving spatial dimensions, while doubling the channel dimension. Next 2 blocks use dilated convolution, which keeps spatial dimensions constant. Block 7-10 upsamples the compressed image and bring it back to original dimension. All the kernels used here were $\mathbf{3 \times 3}$ in size. Fully convolutional architecture ensures the network can work with images of any size.

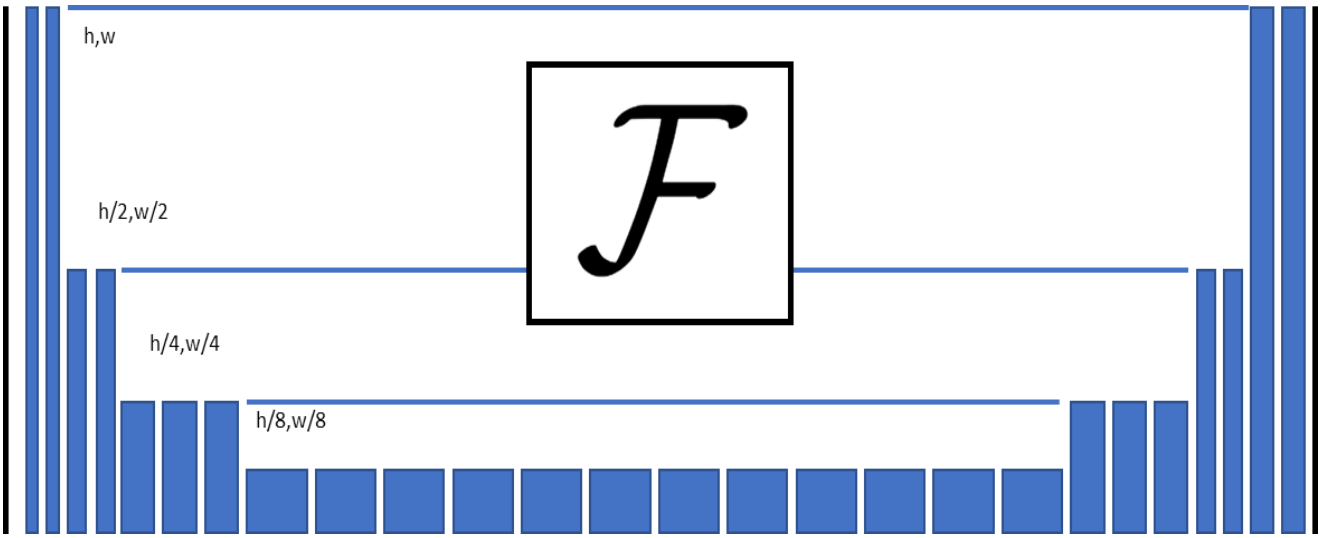

Figure 2: Network Architecture Used By Algorithm in [2]

3.2 IMAGE STYLIZATION

Goal of a neural style transfer algorithm is to generate an image that combines content from one image and style from another. In this work, we also focus on abstract styles as defined by art historians. This means instead of capturing the style of just one painting, we automatically learn visual characteristics of a particular style by looking at hundreds/thousands paintings of that style and combine it with content image. The algorithm used here for single style stylization was proposed by [4], and the one for abstract style by [5]. During training, Places Dataset [6] was used for content and Wikiart dataset [7] for style.

3.2.1 Single Style Transfer

A trained convolutional neural network (trained on image classification) is used to extract features from images. So, an image of initial size (W, H, C) becomes (w, h, c) after passing through some layer of CNN.

This projects both style and content image to feature space using CNN. Content feature is then transformed to have identity covariance matrix for channels (whitening transform). When D_c, E_c are respectively the diagonal eigenvalue matrix and eigenvectors of covariance matrix of content feature, transformed content feature is:

$$\hat{f}_c = E_c \, D_c^{-\frac{1}{2}} \, E_c^{\top} \, f_c$$

Content features further undergoes another transformation to have same covariance as style image, known as (Coloring transform). In equation below D_s, E_s are respectively the diagonal eigenvalue matrix and eigenvectors of covariance matrix of style feature. Our final transformed feature is:

$$\hat{f_{cs}} = E_s \, D_s^{\frac{1}{2}} \, E_s^{\top} \, \hat{f_c}$$

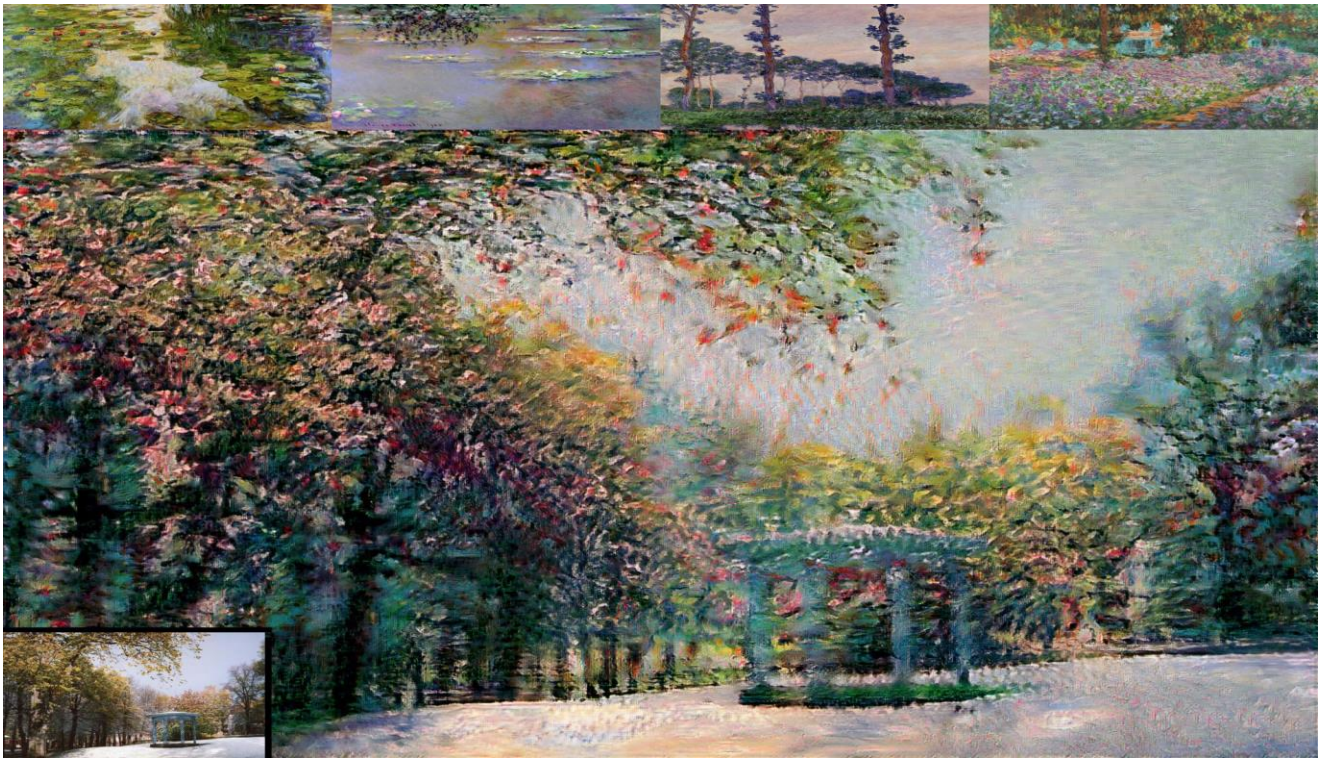

Figure 3: Example of abstract style transfer. Style here is artworks of Claude Monet (4 representative paintings at top. Content image is in lower left corner). [5]

Lastly this final content features are passed through a decoder to get stylized output. To improve performance, this process is repeated 5 times using output image as new content image, and a lower level CNN layer for feature extraction.

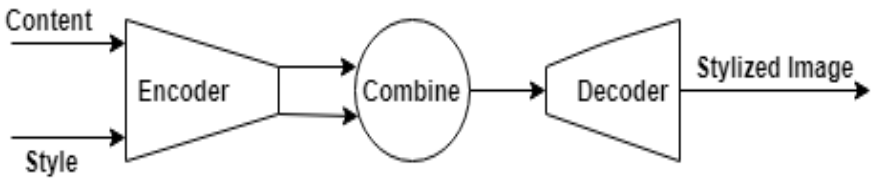

Figure 4 : High level architecture of single-style transfer

During training, we use a pre-trained CNN for Image classification task to project images into feature space. Decoder training had two objectives. First it simply trained to work as autoencoder, i.e. models capable of taking a point in feature space and projecting it back into image space. Second objective wanted to compare perceptual similarity that compared instead of comparing input image with output image, instead compared their high level, low dimensional features. Precisely, When I_o, I_i are respectively our input and decoded image and Φ encoder function, our training objective is:

$$L = \|I_o - I_i\|_2^2 + \lambda \|\Phi(I_o) - \Phi(I_i)\|_2^2$$

λ is weight to control relative importance of these two objectives.

3.2.2 Abstract Style Transfer

To understand common characteristics of a particular style, this algorithm [5] employs generative adversarial network. Given a group of samples (group of similarly stylized images in this case), GAN tries to learn hidden distribution that generated them. GAN have two components, generator and discriminator (D). Generator itself can be divided into 2 components: Encoder (E) and Generative Decoder (G). When the generator produces a sample, discriminator returns the probability of this sample belonging in original distribution i.e. probability of stylized image really belonging to the style shared by group of stylized images. Generator tries to maximize this probability and progressively gets better at producing right samples.

During training for a particular abstract style, thousands of content images are stylized by the generator. These stylized images are checked against original style images by discriminator. Generator takes the feedback from discriminator to adjust its parameters and produce better stylization in next iteration. Please note this algorithm requires one whole generator network for learning just one abstract style. Formally, given X as our content dataset and Y as the style dataset, loss function to that generator tries to minimize and discriminator tries to maximize is:

$$\mathcal{L}_D(E, G, D) = \mathop{\mathbb{E}}_{y \sim p_Y(y)} [\log D(y)] + \mathop{\mathbb{E}}_{x \sim p_X(x)} [\log (1 - D(G(E(x))))]$$

First part of the equation can be understood as expected probability of calling “real samples real” and 2nd part as calling “fake ones fake”.

The generator part of the network resembles the architecture shown above. Difference is this network is now appended with one extra “discriminator” model. The overall architecture is shown below:

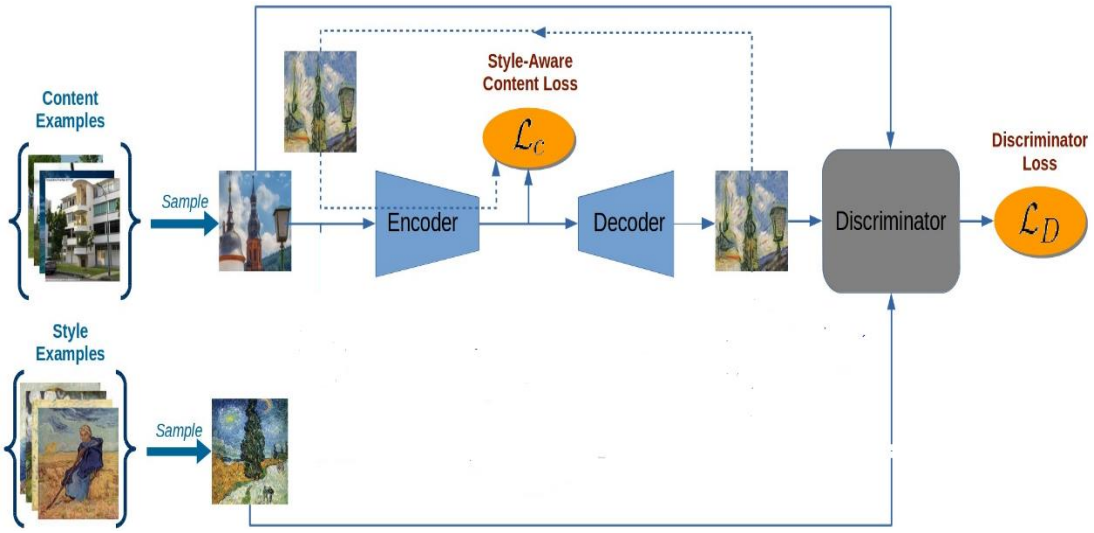

Figure 5: Architecture of Abstract Style Transfer Network (Image taken from original paper, slightly edited for clarity)

4 ARCHITECTURAL DESIGN

4.1 OVERALL ARCHITECTURE

DeepPaint is designed to be easily extensible. Although it currently provides two functionalities, we aim to extend DeepPaint with several other promising deep learning-based computer vision algorithms in future e.g. Image Inpainting, Video stylization, Sketch to Image etc.

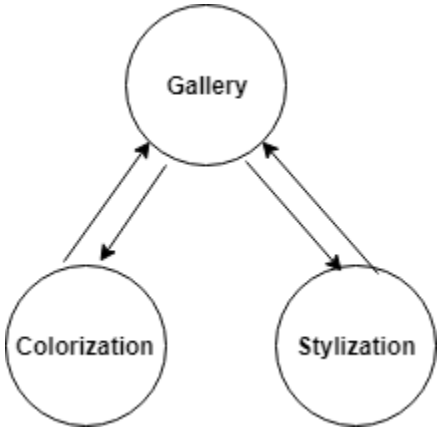

Figure 6: Core Components and their dependence.

At the core of whole system and all components is ***Network*** abstract class. This represents a neural network that forms the basis of all the other components and their functionalities. Another entity common to all components is a ***Gallery*** class, as depicted in above figure. Upload and download methods are used by end users, whereas get and put methods by components.

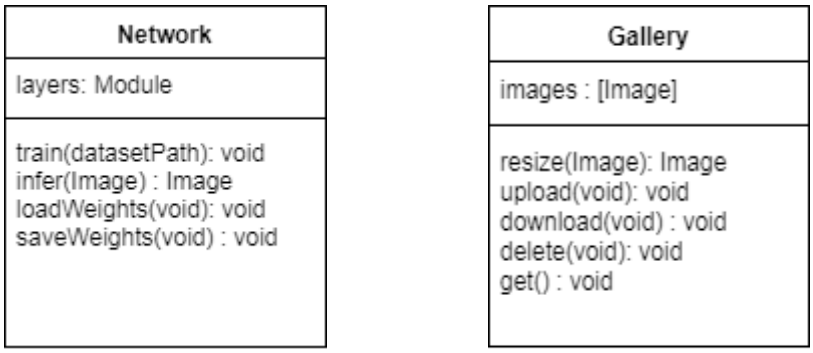

Figure 7: Classes belonging to overall system.

We next briefly talk about design of individual components and rationale behind these designs.

4.2 COLORIZER COMPONENT

At the root of this component is another abstract class ***Colorizer***. Although not explicitly mentioned in usage scenario, apart from the local hints-based colorizer already mentioned, we aim to implement another colorizer based on global statistics. Abstract ***Colorizer*** class defines the common architecture of these two separate networks. Both local hint and global statistics based models extend that architecture to implement their own unique functionalities.

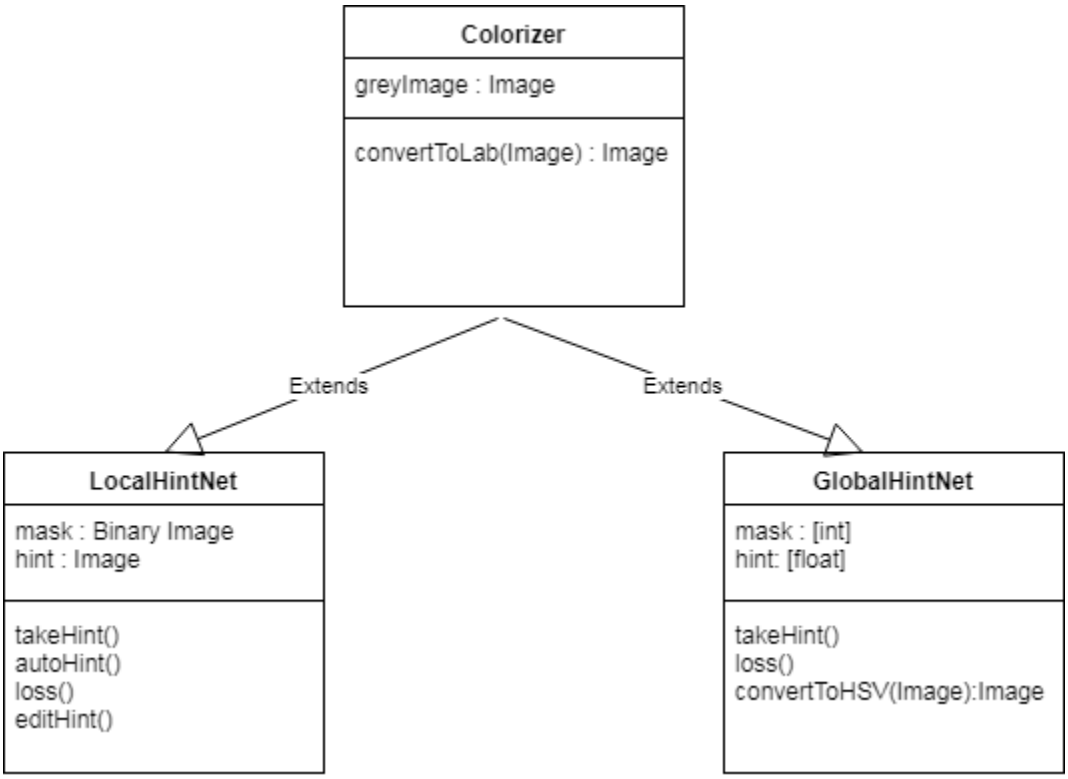

Figure 8: Architecture overview of Colorizer component

Both classes need to work with CIE Lab color space. ***GlobalHintNet*** also needs LSV color space images to compute image saturation. Mask in case of local network refers to points annotated by user as hints, and have same spatial dimension as original image. For global hints network, it refers to saturation value for different quantization bins.

.

4.3 STYLIZER COMPONENT

As mentioned, there are two types of stylization functionalities available to user. Each is implemented by their separate class, and both extends abstract ***Stylizer*** class. For single style transfer, we need a style image. Another method needed is a way to combine encoded features. Abstract style transfer requires training for a group of style images and therefore need path to that dataset. It also implements generator/discriminator methods discussed earlier. Loss functions are overloaded for both classes.

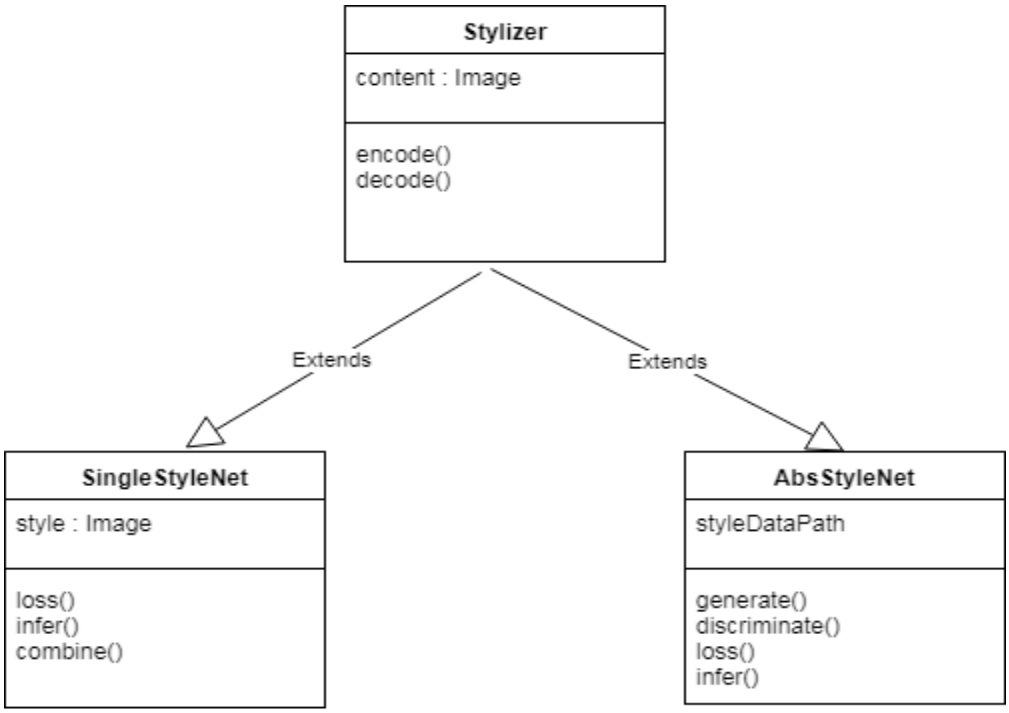

Figure 9: Architectural overview of stylizer component

The final architecture, depicting class dependence can be represented as:

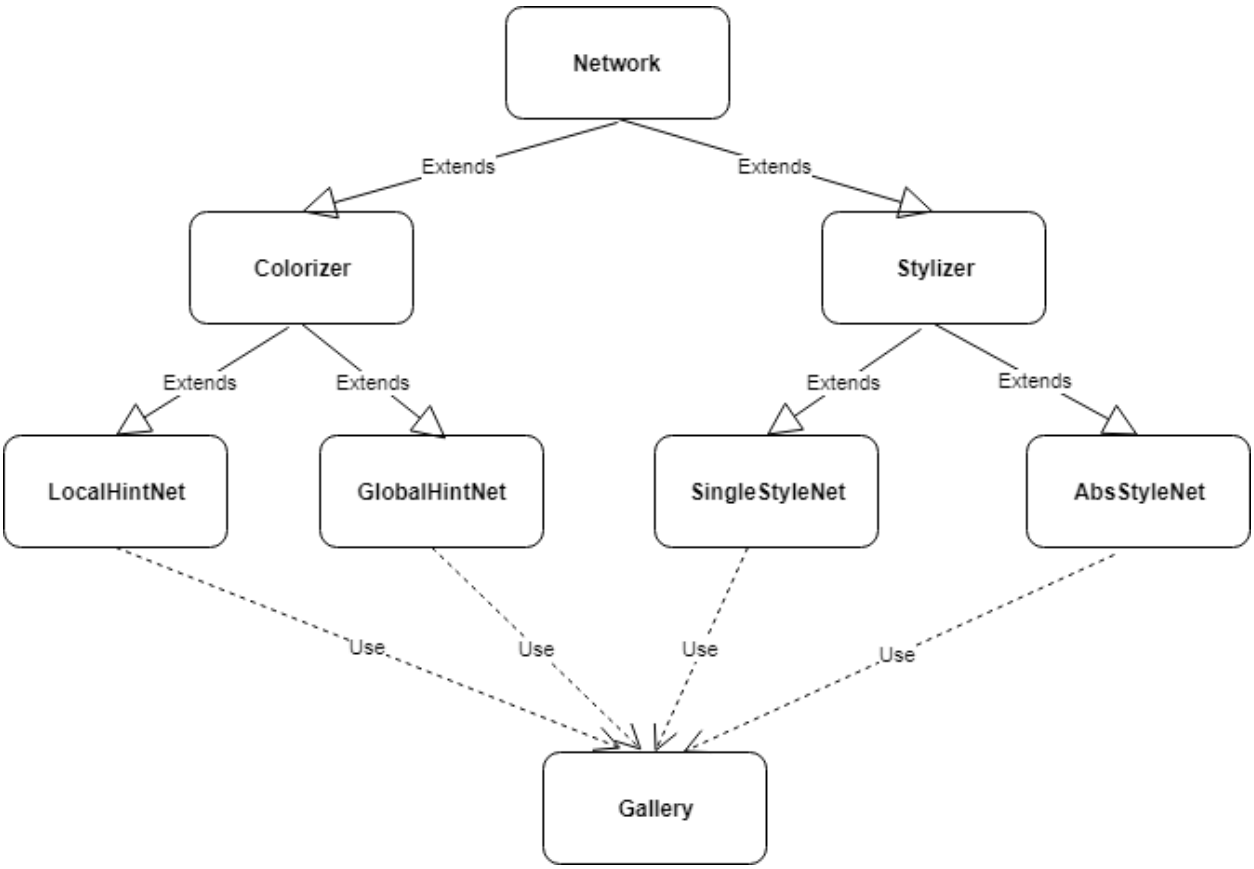

Figure 10: Class relations in overall project

Please note the lack of inter dependence between individual components.

5 IMPLEMENTATION

5.1 DEVELOPMENT ENVIRONMENT

The language used for development is Python. We made use of Jupyter Notebook for rapid development. We used Git (with Github) for version controlling of the program. The complete code can be found at: https://github.com/Redoykhan555/DeepPaint.

One of the most important libraries used is Pytorch. This library allows gradient-based programming, especially deep learning related tools, and was chosen due to its simplicity compared to its main rival TensorFlow. For front end GUI, we used PyQt5.

5.2 CODE OVERVIEW

We structure this section based on the files and folders of the main program.

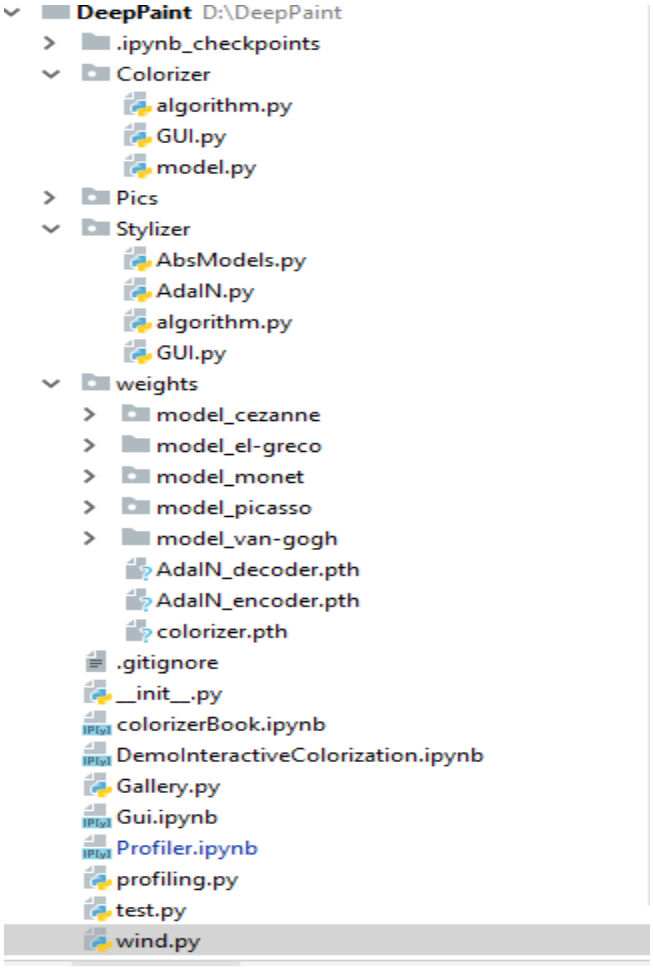

Figure 11: Source code structure, taken from PyCharm IDE.

Colorizer: This folder contains entire colorizer module, both machine learning code and GUI. It only exposes the GUI, abstracting away the algorithmic details.

***model.py*:** This file simply contains the network architecture as described in Methodology section.

***algorithm.py*:** This file import model and handles colorization related tasks like preprocessing images, hints etc. "`lab2rgb_transpose`" function converts a LAB space image to RGB. "`run`" function normalizes the inputs to model and then forwards through the model to actually obtain the output. "`put_point`" function converts RGB user hints to LAB and puts them in mask. "`colorize`" calls them all to provide the colorization function and is the only function exposed to outside.

GUI.py: As the name implies, this builds graphical interface and connects them with underlying algorithms. This will become a Tab in main window.

Stylizer: This folder contains entire stylizer module. Like colorizer, it also only exposes GUI.

AbsModels.py: This contains both architecture and related functionalities for abstract style transfer. It loads the model weights during module initialization. "`stylizeAbstract`" function handles image preprocessing, normalization and then post process the output.

AdaIN.py: This contains everything related to AdaIN, the algorithm used for single style transfer. "`calc_mean_std`" is a helper function to "`adaptive_instance_normalization`", which combines extracted features. "`stylizeAdaIN`" is the main public function that takes two images as input, along with stylization strength, to produce stylized output.

algorithm.py: It simply unifies different stylization algorithms and provide a simple interface to ***GUI.py***. It has a single function "stylize", that depending on input, either calls "`stylizeAdaIN`" or "`stylizeAbstract`" under the hood.

Pics: This provides some sample paintings and images to easily test or get started with DeepPaint.

Weights: This contains all the trained weights for all the models used throughout the project.

Gallery.py: This file provides gallery related functionalities and GUI.

wind.py: This brings together all GUI.py as tabs in a main window, along with the gallery.

profiler.py: To monitor resource usage and collect statistics.

test.py: This contains all the unit test suits to be discussed in detail in following section..

Files of "`*.pynb`" format are Jupyter notebooks, to allow rapid development. They will not be part of final delivered product.

One of our primary design goals was to decouple algorithms from GUI as much as possible, mainly to be able to provide multiple types of user facing clients among other reasons.

5.3 How Open Source Implementations were used

All the algorithms implemented here have open source implementations. Colorizer algorithm have an official Pytorch implementation. But we simply took the model architecture from that repository and all other codes related to training and inference were rewritten. We believe the code structure, GUI etc. all to be sufficiently different. This is especially evident in the fact that our inference and GUI related code combined is around 1/2 in size (in LOC) of official implementation.

As for style transfer, there are right now 2 algorithms present, with plans for adding more. Abstract style transfer had original implementation in Tensorflow, a framework that is considered to have different paradigm than Pytorch. We re-implemented the inference portion of the code, and trained weights were taken from official repository. We'll soon complete the training part. For AdaIN, like colorizer, we consulted official repository to clarify ambiguities, and Decoder part of the model architecture was taken directly. All other code is our own.

6 TESTING

6.1 UNIT TESTING

One limitation of unit testing in DeepPaint is that outputs are mostly subjective to judge. Meaning there is no simple way to define “expected output” as is the case with for example a sort function. So, our aim here was to reveal functions inability to handle wide range of inputs in the form of exceptions.

Unit testing was carried on only algorithmic parts of modules i.e. GUI parts were excluded. We used Python’s built in “unittest” module for this task. A similar group of tests for testing a particular module are organized in subclass of “TestCase” of “unittest” module, where each unit of test are methods of that class. For our testing, we wrote two different TestCase, “TestColorizerAlgo” for Colorizer and “TestStylizerAlgo” for Stylizer module.

The individual unit tests were:

`TestColorizerAlgo`

- `test_colorize_size`: For testing images of different sizes.
- `test_colorize_points`: Simulated different number of user hints, corner cases include no hint, multiple hints for same pixel, all pixels hinted etc.
- `test_colorize_colors`: Simulated different types of colors and different radius in hints.

`TestStylizerAlgo`

- `test_AdaIN`: This function tests single style transfer’s “stylizeAdaIN” function.
- `testAbstract`: This function tested “stylizeAbstract” function of abstract style transfer.

In both cases, care was taken to ensure wide variety of images were passed as input.

6.2 PROFILING

One of our key concerns throughout development was the performance of the algorithms. Our aim was to ensure real time performance at least for small to medium sized paintings. In charts to follow, we present the performance of 3 algorithms depending on size:

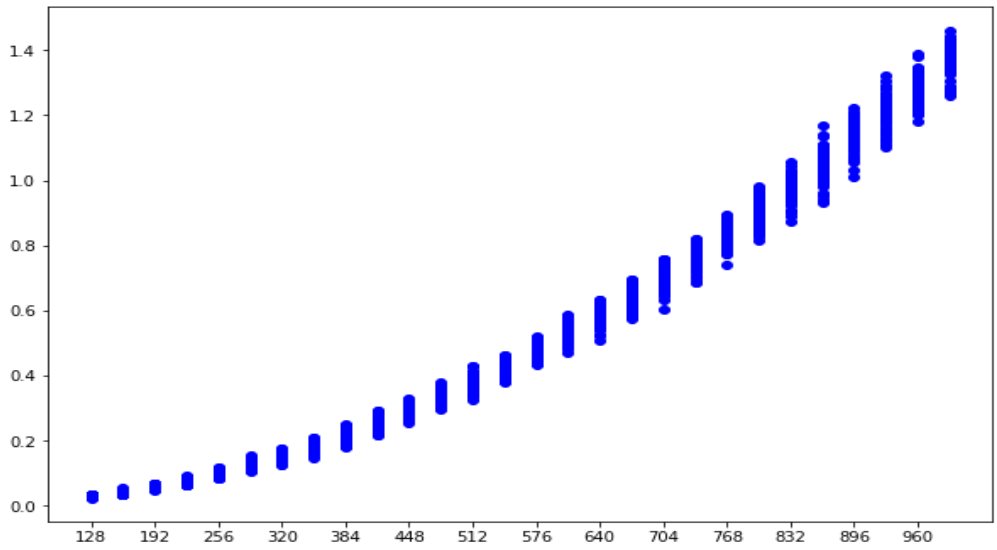

Figure 12: Time (Y-axis) vs Size for colorization

The above plots time taken in seconds vs size of one dimension of squared paintings. As can be seen, the plot isn't linear, so the colorizer might take too long for extremely large paintings.

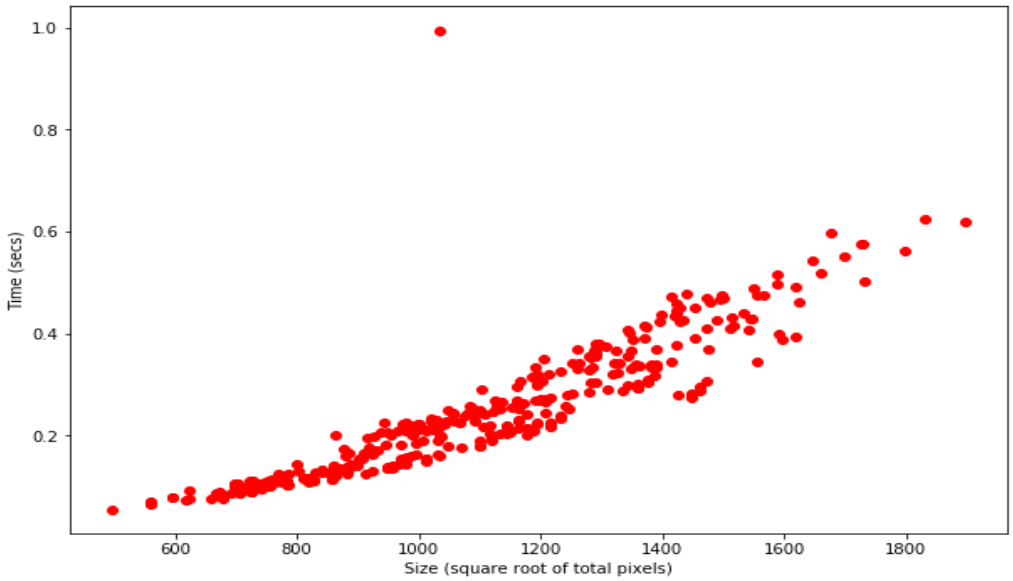

Figure 13: Time taken for single style transfer.

75th percentile of time taken was 0.3135 seconds, meaning most of images can be style transferred in fraction of a second. Maximum time was .99 seconds, still less than a second.

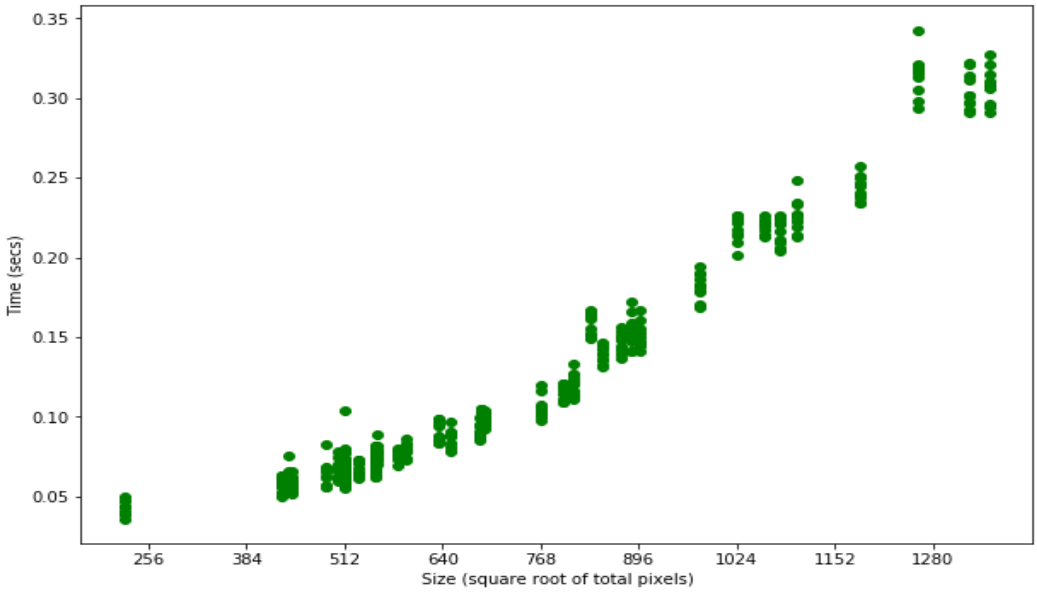

Figure 14: Time taken for Abstract Style Transfer

Abstract style transfer requires a just single pass through the pre-trained models. So, they are comparatively faster. On average it took 120 milliseconds, with the biggest painting taking 340 milliseconds.

6.3 Bugs Discovered

Here we list some of the bugs found using both unit testing and exploratory testing:

- Resize Bug in Colorizer. Colorizer model can only work with images of squared size. Test function “test_colorize” of test case “TestColorizerAlgo” revealed failure in non-square sized images.
- High resolution images were found to crash due to memory limitation.
- Pressing Undo on a grey scale painting without any hints would result in program crash.

7 User Manual

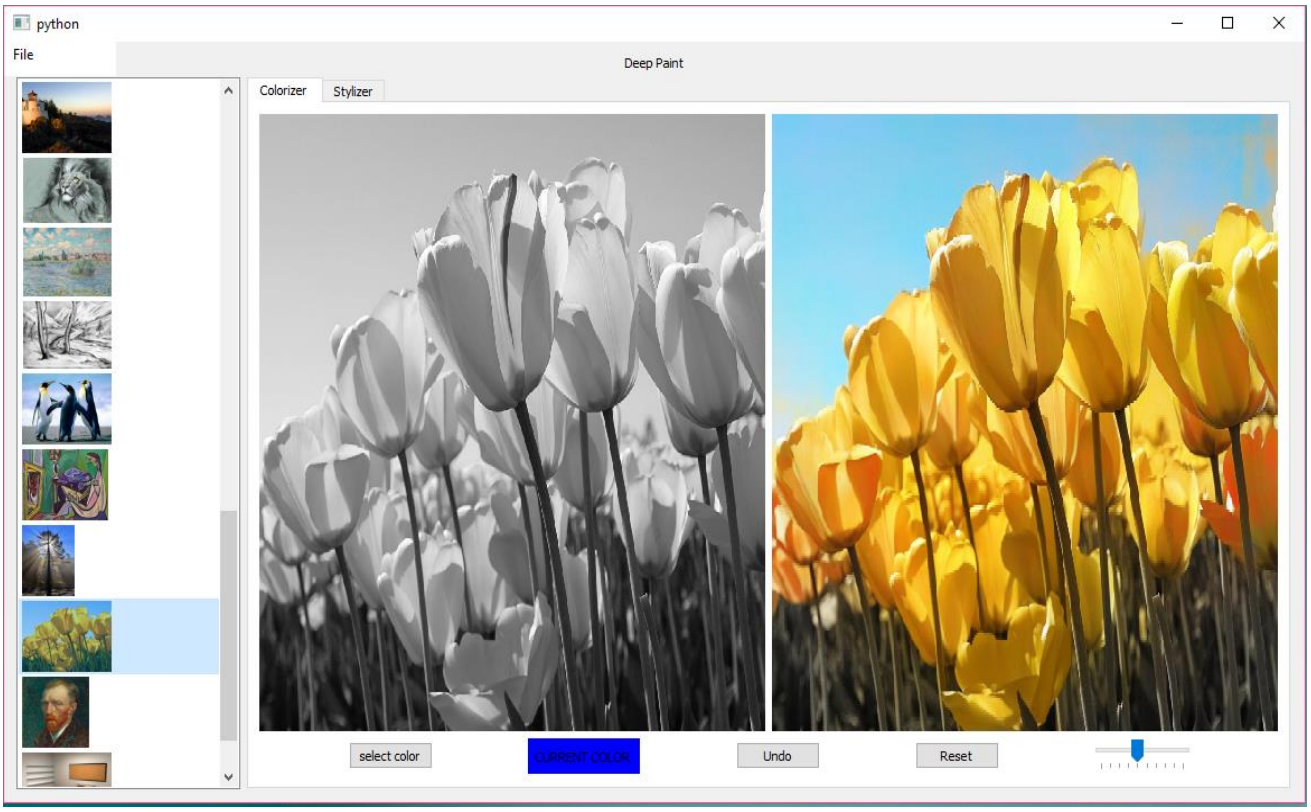

Figure 15: Main window of DeepPaint

7.1 Installation

This program is divided into server and client. Setting up a single server would require:

- A CUDA compute capability >= 6.1 supported NVidia graphics card.
- 8GB of RAM
- Windows 10
- NVidia CUDA 9.0
- NVidia cuDNN 7.1
- Python >=3.6

When above requirements are met, please run "`pip install -r requirements.txt`" to complete installation. To start a server run "Python server.py", which will open up the server in port 1539. You can change default port by using -p option from command line.

The client program has no requirement. Simply type DeepPaint.exe from command line and main program window will appear. Use "-ip" and "-p" port to specify server's IP and port respectively.

7.2 Components and How to use them

There are 3 main components of this program. They are:

1. Gallery: To manage all paintings.
2. Colorizer: To automatically color a painting or guide the colorization AI through point hints.
3. Stylizer: To stylize a painting, using either:
    a. A reference style image
    b. Established art styles like Expressionism etc. (Called Abstract Style from now on.)

We now discuss how to use these components in detail:

7.2.1 Gallery

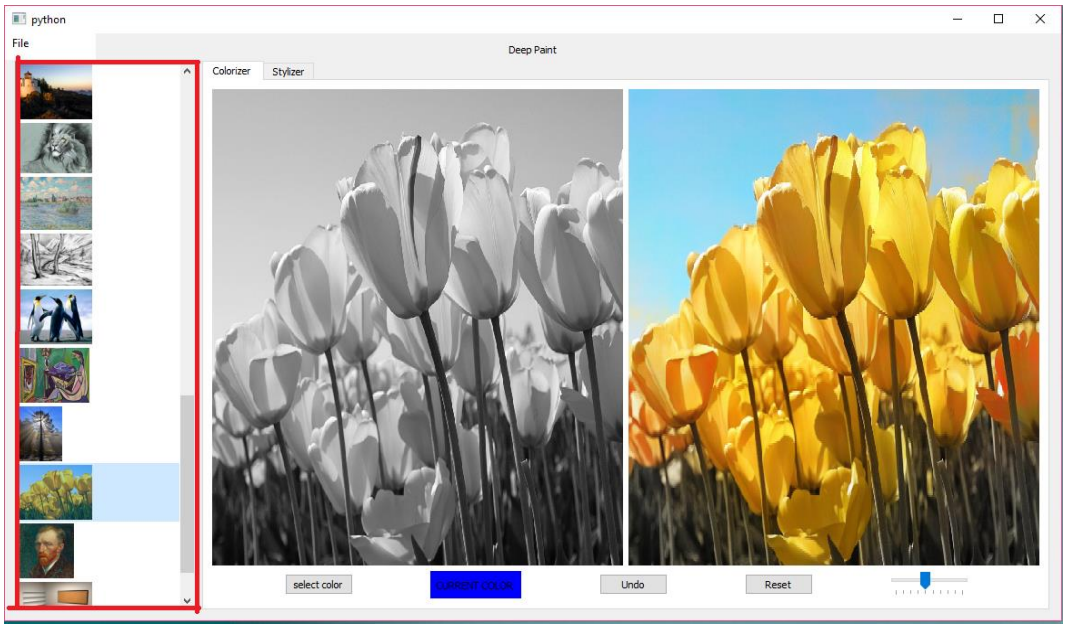

Figure 16: The Gallery (Outlined in Red border)

The gallery can be thought of as a shelf containing all the finished or unfinished artworks. This can be used effectively as version controlling a painting at different stages of its refinement.

7.2.1.1 Uploading Image

To upload an image, please click the “File” menu in the top left corner on the menu bar. You can either choose to upload a file, or an entire folder containing images from the options. After file or folder has been selected, the image(s) will automatically appear on the gallery.

7.2.1.2 Moving Paintings for Components

Both algorithms require one or more paintings as input. This process is extremely simple through the use of mouse dragging. You can simply drag a painting from gallery to a

component's input box. The output paintings also can be dragged into gallery, hold right mouse button for this dragging.

7.2.1.3 Other Functionalities

These functions can be accessed by right clicking on a painting in the gallery. They are:

- Downloading paintings. A dialog window will appear asking you to select a destination file path for the painting to be downloaded.
- Deleting paintings from gallery. Please be careful as this action cannot be reversed.
- Resizing a painting. A dialog box will appear asking for newer height and width. The original sized painting will not be deleted.

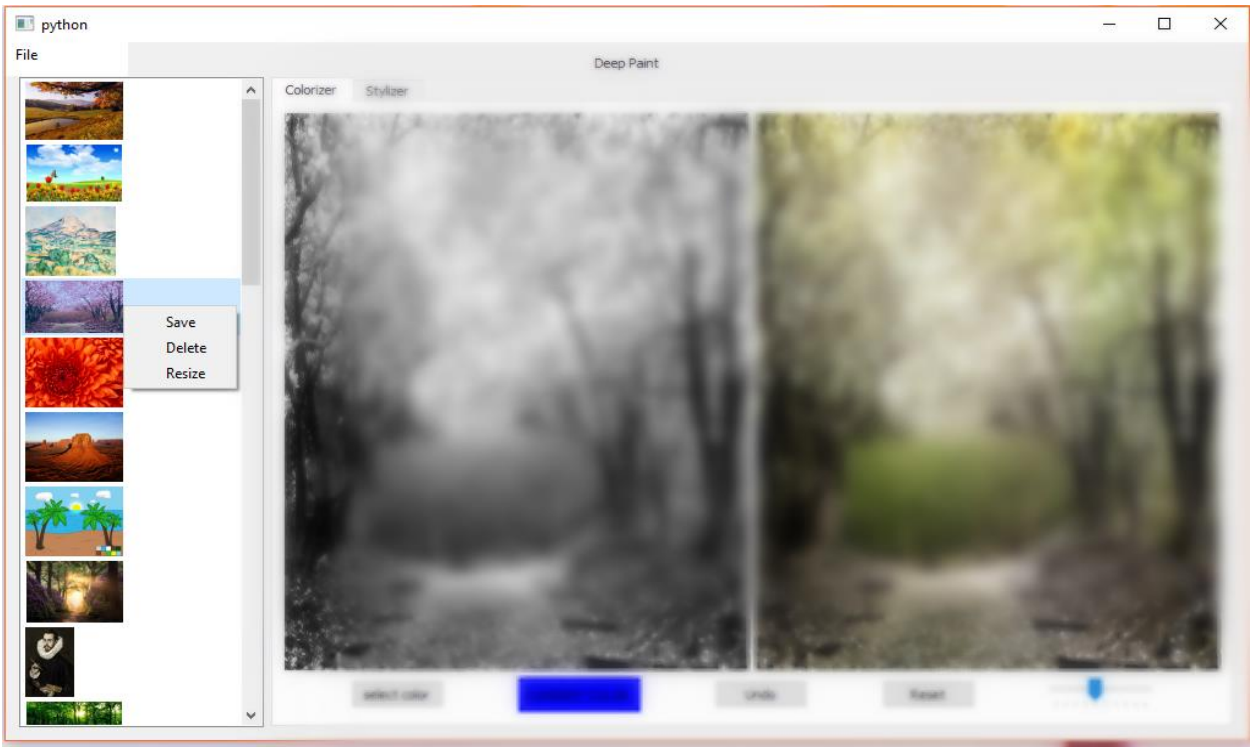

Figure 17: Context menu in Gallery evoked through right click

While working with painting, please be careful to not over-burden the gallery with too many paintings due to memory constraint. Please also remember the size of individual paintings for mental calculation.

7.2.2 Colorizer

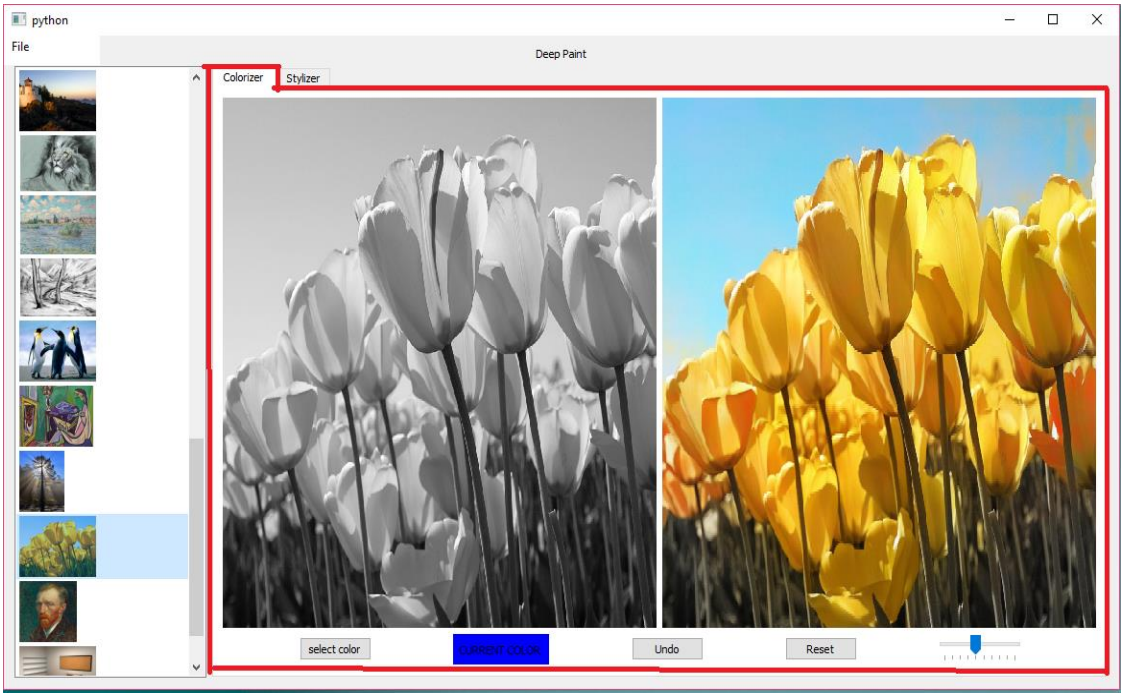

Figure 18: Colorizer Component (Pictured above is automatic colorization)

Upload the painting of interest to gallery first if it already isn't there. Now click on the "colorizer" tab. You will see two empty boxes. Now simply drag that painting to the left empty box. You don't have to do anything else. The program will detect new painting has been placed, automatically colorize it and put the output in right sided Output Box. It won't complain against already colored paintings. The current colors will be thrown away and it'll be turned to a grey-scale image.

You can provide hints in the form of colored points to guide this colorization. To achieve this, simply click at any point in the grey-scale image to plant a hint. Below the input/output boxes is a panel containing two options for this task. To choose a particular color, click select color button. The selected color will be previewed in the label right next to this button. There is also a slider, which can be used to control the radius of points.

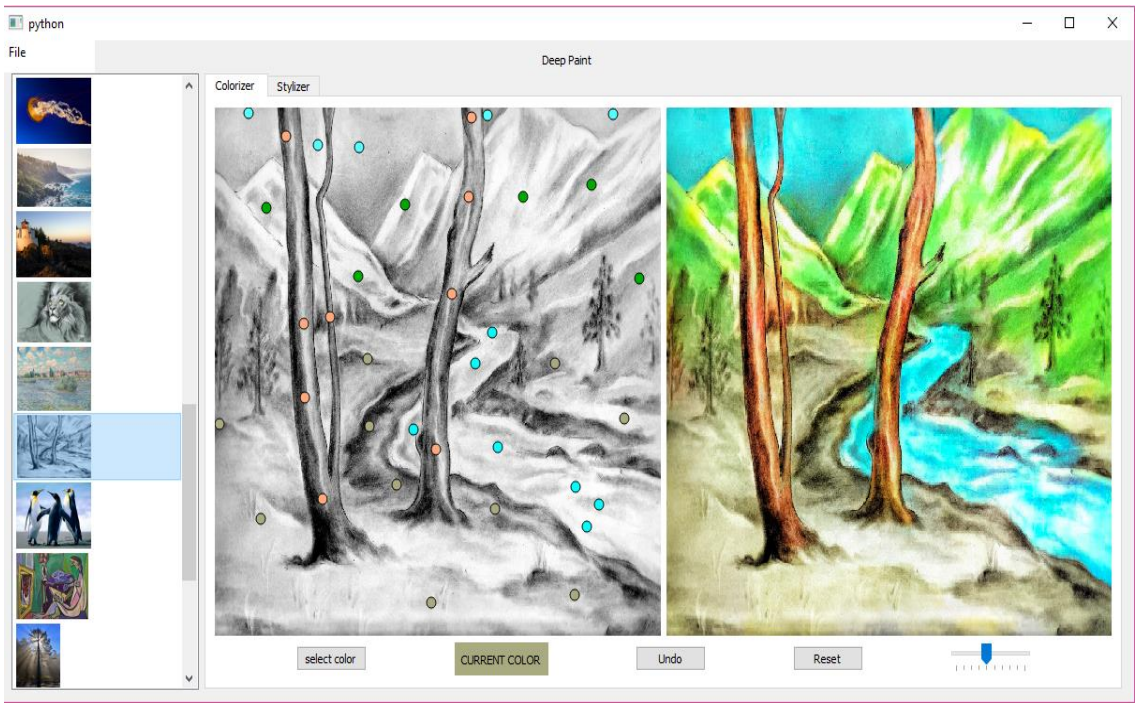

Figure 19: Coloring a black & white sketch using Hints

Another two important buttons here are “Undo” and “Reset”. As their names imply, they can be used to undo last planted hint or remove all hints from the current painting. Both will result in updated colorization in output box.

Please note that colorizer can now only work with fixed size paintings (All paintings are automatically converted to 512X512 before colorization takes place). You can change the size of colorized image by bringing it to gallery and accessing Resize function from there. Please also note that neither automatic nor hint-guided colorization is matured at this point and can be lacking at quality.

7.2.3 Stylizer

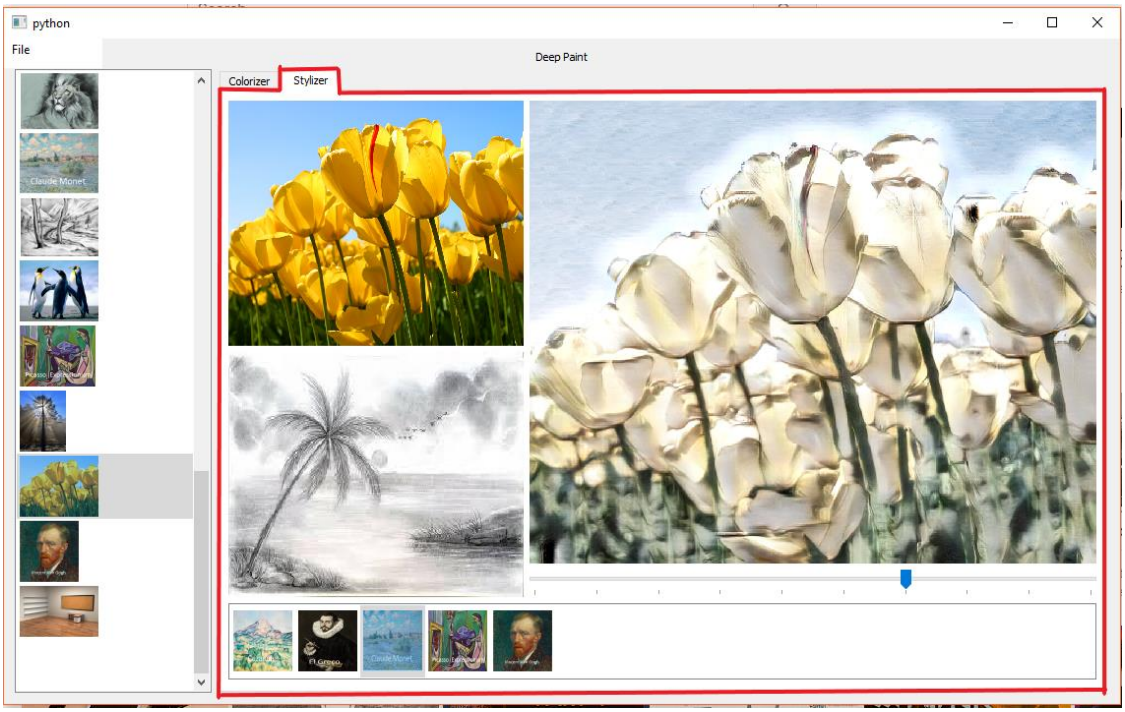

Figure 20: Stylization Component (Single Style Transfer)

Stylizer takes a “Content” painting and stylizes it along the style of a reference painting or an abstract style. Once you’ve opened Stylizer tab, you can populate both the Content image box (top left) and style box (bottom left) using paintings from gallery. This is doing single style transfer.

A slider right below the output box can be used to control the “stylization strength”. Higher the stylization strength, more powerful will be the impact of style painting in final stylized image. A stylization strength of 6 can be roughly understood as 60% stylized image and 40% original image. This slider usually works in real time, but some bigger sized paintings and expensive models may have lags.

Figure 21: Effect of stylization strength on stylizer

For abstract style transfer, there is set of paintings below input/output boxes. In the same way as single style stylization, they can be dragged into style box for abstract stylization. Controlling style strength for abstract style is not yet available.

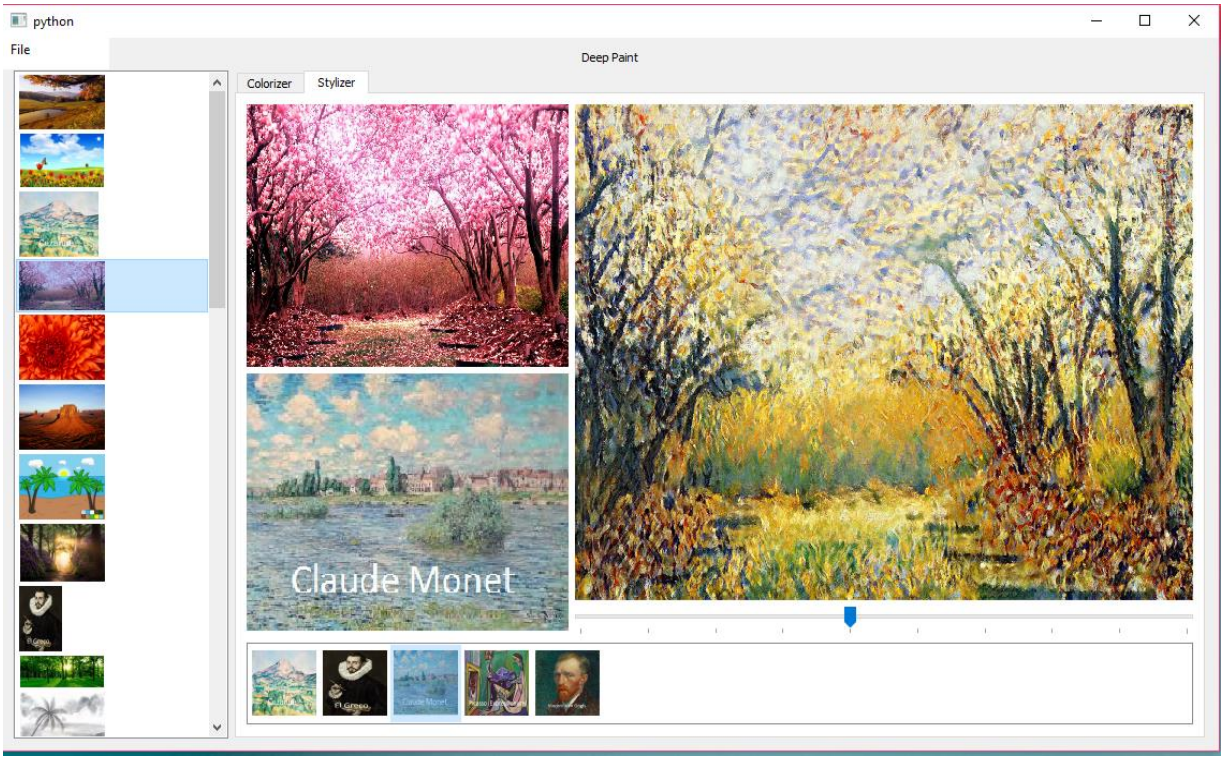

Figure 22: Abstract Style Transfer (Landscape style Of Claude Monet).

In both cases the stylization process works in real time for paintings below 1080p.

7.3 Guidelines for Effective Use

Style transfer effectively destroys the color of content image, so first colorizing the painting of interest and then stylizing would destroy the effect of colorizing. So it'd be better to first stylize then colorize.

Alternatively, you can choose to colorize possible stylizing image, thus creating a new version of it in gallery and then stylize using the new stylized image. This can be helpful for say if you want to change apparent day of time in painting from afternoon to sunset by replacing high presence of yellow of afternoon by red of sunset. But please remember stylization algorithms only cares about the colors present, not their spatial position in painting. So there is no need to do careful colorization in above example.

Briefly, some other things you can do to bring the best out of DeepPaint include:

- Reduce size of painting of interest and work with reduced size. This will allow you to try more combination of colorizations and stylizations as both can handle small to medium paintings in real time. Once you've found desired combination, you can apply it to bigger, original painting.
- Please remember algorithms used here are data driven. So performance can vary between paintings but more importantly, can be unpredictable. The colorization component specially may require some patience.
- Colorization component have been found to work well with lighter and common colors. Since the algorithm was trained with real data, it might struggle to work with little too creative application of color, for example a green sky.
- The style strength slider is pretty important for single style transfer, as default value might not always lead to best possible outcome.

8 CONCLUSION

Neural style transfer and image colorization are two of the most important fields of computer vision. And both has been revolutionized by deep learning. In this project, we hoped to utilize the strength of deep learning to help bring these two powerful algorithms into the hands of end users. Although this project specifically targets amateur paintings, we believe this tool is general enough to be used in other contexts, or can be extended easily.

There are lots of ways this tool can be improved or extended. Due to the rapidly growing nature of related fields, lots of improvements in fact surfaced in academic community during our development time. For example, using multi-task networks, instead of using separate architectures for each tasks, we believe these algorithms can be made to share at least part of their models, thereby hugely reducing memory and space requirement. Some components, especially colorizer have lower than expected performance, which can either be corrected by borrowing ideas from recent papers, or completely replaced with newer algorithms. We also hope to bring the entire program to web in near future, allowing end users to sidestep the comparatively hard task of server installation.

In this development process, we were fortunate to work with some cutting edge computer vision techniques and algorithms. Colorization and stylization simultaneously proves the amazing strength of recent computer vision technologies, and their weaknesses. On one hand, the scope and goals of this work appears pretty broad, which gave us a unique opportunity to familiarize ourselves with vast field of computer vision. On the other hand these algorithms shares lots of underlying concepts, giving us a somewhat unifying view of the field.

Personally, this has already been a great learning experience, and hopefully will continue to be so. In future, we plan to tackle some of the current issues mentioned above, and hope to push both our product and these fields in general forward.

9 REFERENCES

[1] Wikiart.org, "Wikiart".

[2] R. ZHANG, J.-Y. ZHU and A. EFROS, "Real-Time User-Guided Image Colorization with Learned Deep Priors," 2017.

[3] A. Krizhevsky, l. Sutskever and G. Hinton, "Imagenet classification with deep convolutional neural networks," 2012.

[4] Y. Li and M.-H. Yang, "Universal Style Transfer via Feature Transforms," in *NIPS*, 2017.

[5] A. Sanakoyeu, D. Kotovenko, S. Lang and B. Ommer, "A Style-Aware Content Loss for Real-time HD Style Transfer," in *ECCV*, 2018.

[6] Z. B, L. A, J. T. A. Xiao and Oliva, "Learning deep features for scene recognition using places database," in *NIPS*, 2014.

[7] S. Karayev, M. Trentacoste, H. A. Han, D. T. A. and Hertzmann, "Recognizing image style," in *Arxiv Preprint*, 2013.