Imitating Artistic Style on Automatically Curated and Shaded Sketches

Jigyasa Grover
grover.jigyasa1@gmail.com

Richa Banker
richa.7b@gmail.com

Abstract

Owing to their ability to reproduce and model non-linear processes, neural networks have found many applications in a wide range of disciplines. Of late, realistic image generation and imitating artistic style and content of fine art is garnering attention for detailed analysis and finding application in diverse areas of digital entertainment. In this project, initially we curate line-art sketch from the given Manga (Japanese Comics) characters' image using adaptive thresholding and then deploy Generative Adversarial Network to automatically shade the sketch, where compatible colors are learnt at pixel level from a large collection of images. Later in our experiment, we use a set of sample artistic images to transfer the style onto the shaded images previously generated to create artistic images of high perceptual quality using VGG-19 pre-trained model. The system uses neural representations to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images. As a result we were able to synthesize shaded images with a maximum of 90% (SIFT) similarity and visually appealing artistic imitations of the same.

1. Introduction

Human mind is particularly sharp and possesses an appreciable level of cognitive capability to comprehend black-and-white line-arts, shade these sketches with the choicest of colors and also imitate artistic style (*e.g.* brush stroke, etc.) from an artist's previous work and fashion a visually appealing artwork by composing a complex interplay between the content and style of an image. With the advent of Digital Art, which has been gaining immense popularity, especially in entertainment (animation) industry, it has become crucial for the machines to do the same thus axing human effort and time spent, along with augmenting the quality of the output. Nonetheless, automating this process has a multitude of practical applications: the production time of asset-heavy creations such as computer games and digital comics can be cut down drastically.

Based on the recent success of biologically inspired vision models: *Deep Neural Networks*, which have been known to demonstrate near-human performance in other key areas of visual perception such as object and face detection, we have worked on fabricating an artificial system based on a Deep Neural Network to curate line art, shade the sketch and further imitate artistic style which can be regarded as an integral image synthesis problem.

2. Related Work

In this particular field of image synthesis, previous work have proposed non-parametric models [1, 2] which matched the sketch to a database of existing image fragments. In recent times, deep neural networks [3, 4] have emerged as a perceptive methodology for image synthesis which can generate detailed images of the real-world. Though past work have also shown the images generated to be blurry, noisy and wobbly [5] and not as photo-realistic owing to missing of many sharp details. Also, the output of the synthesis network is not fully controlled as the generator samples from a random vector in low-dimension and the model is substantially independent.

Contemporary approaches have explored the applicability of controllable deep synthesis, example in super resolution problem [6], semantic labels tagging for objects [7] and gray-scale image colorization [8]. These works have helped in the sketch-to-image problem[9], where the control signals are relatively sparse, thus forming a more more ill-posed problem than gray-scale image colorization. Hence it is required for a model to learn and synthesize images from details not contained in the input, and he network should also learn high frequency textures of the image details not only for the scene elements but also high-level image styles.

For the latter part of imitating artistic style, the problem is regarded as a texture transfer problem where the goal is to obtain the texture from a source image while constraining the texture synthesis to preserve the semantic content of the target image. Similar to the shading problem, this also has been previously handled by several non parametric methods using different ways to preserve structure of the target image. For instance, Efros and Freeman [10] introduced a correspondence map that included features of the target image such as image intensity to constrain the texture syn-

thesis procedure. Hertzman et al. [11] used image analogies to transfer the texture from an already stylized image onto a target image. Ashikhmin [12] focused on transferring the high-frequency texture information while preserving the coarse scale of the target image. This algorithm was further improved by Lee et al. [13] by additionally informing the texture transfer with edge orientation information. Inspired by the power of Convolutional Neural Networks (CNN) the pioneering work of Gatys et al. [14] presented a general solution to transfer of a given artwork to any images automatically. Owing to the success of the above techniques, these find applications in many industrial products (*e.g. Prisma, Ostagram* and *Microsoft Pix*).

3. Dataset

In order to train, validate and test our model, we collected $20,000$ images of different *Manga* (Japanese Comics) charcaters' from *safebooru.org* [15] using a custom script with crawled the anime picture search engine. Owing to the availability of high-quality *manga* images with different painting styles, we chose this data-source as the model gets an opportunity to learn from different painting patterns of images. [16]

Prior research in this domain suggests using lower resolutions for synthetic images [17], hence we resized all the RGB images in 64×64 for a more pragmatic approach followed by a deeper neural network and increased number of constraints to stabilize and ease the training process. We also performed normalization to transform the original intensity values into desirable range of $(0, 255)$.

4. Experimental Setup

For the initial task, the supervised learning model generated a colorful shaded image from a simple line-art working on a pixel level based on the given colored image during the training process. The shaded sketch was then supplied as an input to *ConvNet* which performed another task of image-to-image translation by transferring the style of a sample artistic image onto the shaded *Manga* image.

4.1. Curating Sketches

We started the experiment by creating a rough sketch from the given *Manga* image and cleaning it up into line-art, as the ideal way to train the model is to use pairs of original line-art image and the colored image [16] which was hard to track down. To perform this, we used *OpenCV*'s *Adaptive Threshold* method with a *block size* of 9 for calculating the threshold and a constant of 2 is subtracted from the average value. As can be seen from Figure 1, adaptive thresholding yields results very similar to the original line-art image.

Figure 1: Generating B&W sketch using adaptive thresholding

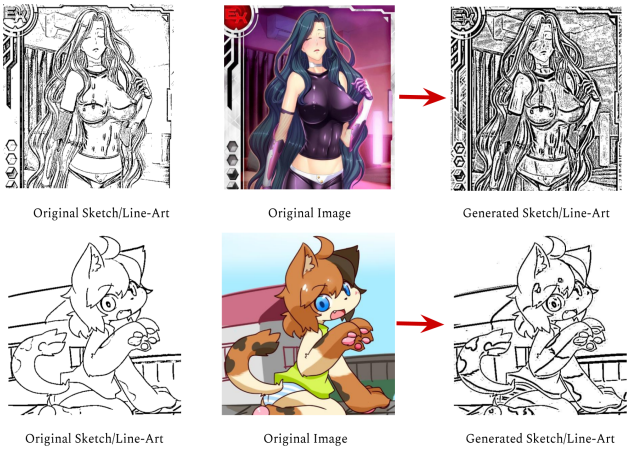
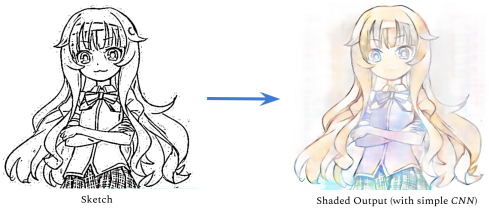


Figure 2: Shading using simple Convolutional Neural Network



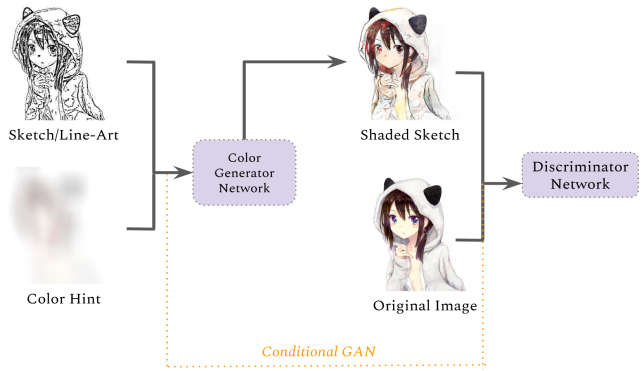
4.2. Automatic Shading

In order to automatically shade the images we steered away from using simple convolutional networks as previous works [18] show that *L2 Loss* is known to create blurry image generations. As can be seen from Figure 2, **baseline** methodology of using simple convolutional network gave shaded images with missing details (*e.g.* eye color, lip shade, etc.), lighter tones and colors spread across the boundaries. Hence, we deployed *Generative Adversarial Network* (GAN) to create images from line-art rather than noise and let the model learn it's own loss function.

Generative Adversarial Networks As suggested in previous research on automatically filling colors in line-art [16], directly using a neural network to learn how to shade a sketch is a very hard problem because finding a good loss function that represents the quality of the image is tough. Hence, for this task we put in use the battle of the generative model G and the discriminative model D to learn inspired by Goodfellow et. all [3] as can be seen in Figure 3.

The framework of GAN contains two neural networks,

Figure 3: Overall working of GAN



the *generator* network $G(z;\theta_g)$ and *discriminator* network $D(x;\theta_d)$. $G(z;\theta_g)$ is the network that estimates the generative distribution $p_g(x)$ over the input data x, where z is noise added to the input sketches. Meanwhile, the network $D(x;\theta_d)$ is used to make a judgment of whether the image is from the real data-set or the one generated by $G(z;\theta_g)$. It maps from point x in data space to a probability. In simpler terms, $G(z;\theta_g)$ directly generates a colored image and the discriminator $D(x;\theta_d)$ network realizes a 2-class classification convolutional neural network [16].

Loss Function To train the two neural networks, we defined *cross entropy* of the classification as the loss function:

$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left(logD(x^{(i)};\theta_d)+log(1-D(G(z^{(i)};\theta_g));\theta_d)\right)$$

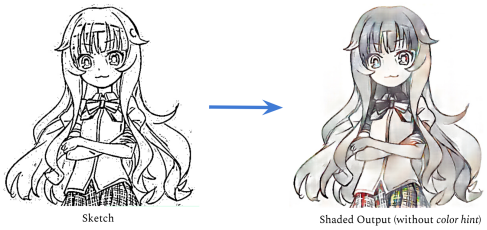
The generator $G(x;\theta_g)$ tries to increase the loss L at each step of the training process in its aim to foll the discriminator, while the discriminator distinguishes the resultant image from the original image and hence it tries to decrease the loss L. Thus, in this adversarial process the discriminator is updated via gradient ascent and the generator is updated via gradient descent.

$$\theta_d = \theta_d + \frac{\partial L}{\partial \theta_d}; \quad \theta_g = \theta_g - \frac{\partial L}{\partial \theta_g}$$

Color Hint via Random White-Out Figure 4 shows the result of the shading when no *color hint* is passed. Since the network is unaware of the color tints it can fill in, it does it's part using the colors guessed using line-art only, *i.e.,* mostly black, white and their tones.

Hence, to have a finer accuracy and avoid color spreading way beyond the outlines caused due to the model being confused when dealing with objects that could be any color

Figure 4: Shading without passing *color hint* to GAN



(such as clothes) unlike skin, we provide the network with a blob of colors as *color hint* which is formed by applying $100px$ radius blur, and pass that as our color hint. But to avoid the network's dependence on the color hint, we performed a contemporary technique [18] of repeatedly (*30 iterations* in our experiment) white-ing out a random 10×10 are of the image and applying the blur and then combining them to form a separate version of *color hint* as in Figure 5.

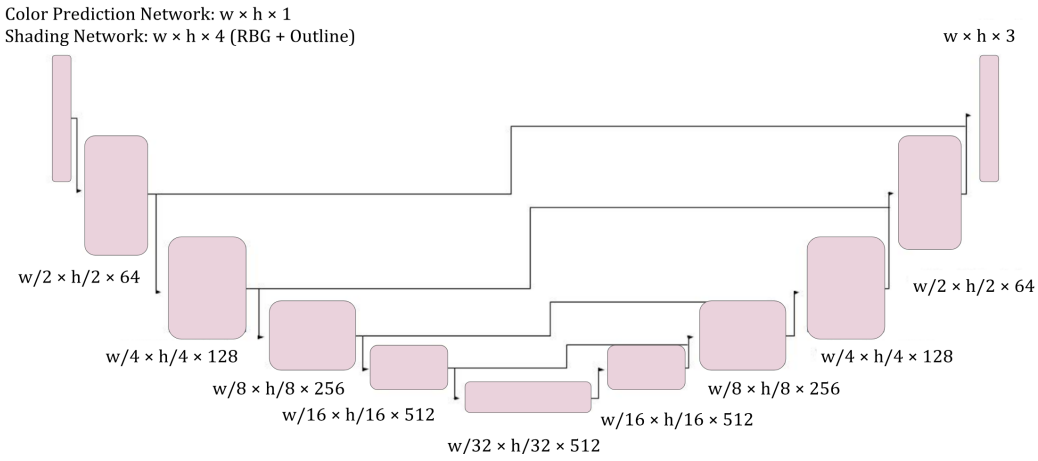
Figure 5: Creating *color hint* to pass to the generative network

Evidently providing color hints seems to work because initially, the network was unaware of hat color tint to fill, hence it became vital to generate good images and thus perform well on simulated color hints. Therefore, corresponding to the inclusion of *color hint*, the framework of GAN changed by concatenating it to both generative and discriminative networks to make judgment conditioned on the color hint as: $G(z, z_{hint};\theta_g)$ and $D(x, z_{hint};\theta_d)$

4.2.1 Architecture

As we require the output shaded image to retain both low-level information (*i.e.*, sketch, outlines) as well as high-level information (*i.e.*, colors, shades, shadows, highlights, etc.), we deploy the *U-Net* [19] encoder-decoder structure as the basis of our conditional generative adversarial network (cGAN) architecture [17]. The two parts of the *U-Net* shaped network (refer Figure 6) are as: the *encoding* phase to the left of down-sampling and the *decoding* phase to the right.

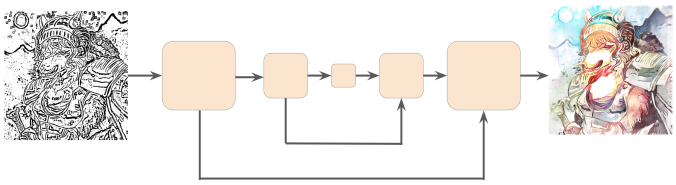
Figure 6: Convolutional network structure.
Arrows represent 5×5 convolutional filters, with a stride of 2.



Fully Convolutional Network Owing to the undertaken image mapping task , the network consisted of *convolutional layers* followed by rectified linear unit (ReLU) and Batch Normalization (BN) layer. As can be seen from Figure 6, we set the convolutional filter size as 5×5 with a stride of 2, hence at each layer the height (h) and the width (w) of the *feature map* decreased by a factor of 2,whereas the number of features doubled at each layer starting from 64. This formed the *encoding* phase of the network. After creating the dense feature map, the *decoding* phase of the network switches to transpose convolutional layers, thereby increasing (doubling) the height (h) and the width (w) of the image and condensing (halving) the features till 64 and eventually coding them into three color (RBG) channel. To enable supplementary flow of information from the encoding to the decoding phase of the network, a direct path from the preceding encoding convolution layer was added to the decoding convolution transpose by concatenating the corresponding convoluted results in encoding phase with the up-sampled results in decoding phase.

Residual Connections Additionally, residual connections were added between each convolutional layer such in *generative* network convolutional layers will operate on the previous layer as well as on all the similar dimenional layers formed previously [18], which can be visualized in Figure 7. Residual Connections allowed the network to have an extended path for heavy processing (such as for facial feature recognition), while still having substantial references to the original input. In our use case, we want the line-art to be present as outlines in the final result as well, hence it becomes an applied methodology to let the information pass through the generative network without passing each layer.

Figure 7: Residual Connections in Generative Network



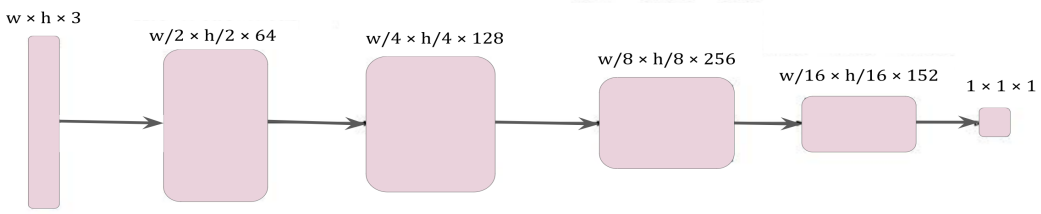
Activation Function For training, we used the *leaky* version of *rectifier* as the *activation function* employing the *leaky rectifier linear unit (LReLU)* because research shows that even though ReLU converts negative values to zeros, and does not change positive values, which leads to a high sparsity of neurons [20], *LRelU* further improves upon the result by allowing a small gradient even when the unit is not *active* [21].

4.2.2 Adversarial Training

As described previously, we make use of the contemporary framework *i.e.*, the *Generative Adversarial Network* (GAN) which learns a custom loss function in the form of a *discriminator* network, that is learned in tandem with the *generator* [22]. The discriminator network maps the image to a scalar and returns a number close to 1 if it believes the image to be real, else a number closer to 0. We constitute the stack of convolution layers as shown in Figure 8 as the discriminator along with one fully-connected layer at the end to map the feature map to the final probability.

At each iteration of training, the *discriminator* network

Figure 8: Discriminator Structure.
Arrows represent 5×5 convolutional filters, while the last represents a fully-connected layer.



was fed with both *real* image and the *generated (shaded)* image along-with their corresponding binary labels. The *generator* was trained to back propagate through the *discriminator* network to convince it to pass the *generated (shaded)* images as *real*. For the *generator* to produce identical looking images, not only visually but also in terms of pixel-wise distance, we introduce a traditional similarity metric of *L2 Distance* as our loss function thus improving stability.

Since the network constitutes of fully convolutional layers which are basically *filters* slid across the entire dimensionality of the image, thus making it reusable regardless of an images dimensions. Thus, despite the training procedure materializing on 64×64, the same network can be used to shade images of different dimensions.

4.2.3 Evaluation metrics

Quantitatively and qualitatively evaluating the standard of the synthesized image is known to be an open and a difficult problem [8]. Apart from visually comparing the generated shaded image and the original image for evaluating realistic image synthesis models, we also used four different metrics to asses the image similarity and quantify the the evaluation.

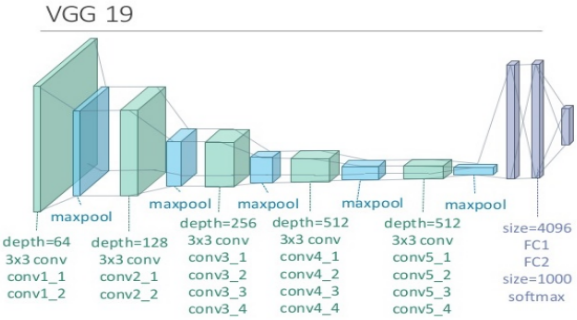
Pixel Similarity This is the most simplest and rudimentary of all evaluation metrics used. Pixel Similarity computes the *L1* norm, *i.e.*, the sum of absolute values of pixel differences.

Structural Similarity Index (SSIM) is a method for measuring the similarity between two images, especially in terms of quality considering one of the defined one to be of *perfect* quality. The SSIM index is a full reference metric; and is designed to improve on traditional methods such as peak signal-to-noise ratio (PSNR) and mean squared error (MSE). SSIM is known to have a near linear correlation with human observer scores, and without invoking the usual, but arbitrary, sigmoid model fitting [23].

Wasserstein Metric (Earth Mover's Distance) Mathematically, Wasserstein or Kantorovich - Rubinstein metric is a distance function defined between probability distributions on a given metric space M. The Earth Mover's Distance (EMD) is widely used to compute distances between the color histograms of two digital images[24], and is also interpreted as the minimum cost of turning one *signature* into the other.

SIFT (Scale-invariant Feature Transform) Similarity Metric calculates the distance between the two images after extracting SIFT features and matching them to evaluate the degree of similarity. It is widely used method in image matching for its good invariance to image translation, rotation and illumination changes [25].

Figure 9: VGG-19 Network



4.3. Imitating Artistic Style

We use the VGG-Network, a Convolutional Neural Network that performs extremely well on common visual object detection. We use the feature space provided by the 16 convolutional and 5 pooling layers of the 19 VGG-Network. This model is publicly available and can be explored in the *caffe* framework.

4.3.1 Content representation

Each layer in the neural network defines a non linear filter bank whose complexity increases with the position of the layer in the network. Hence a given input image $\vec{x}$ is encoded in each layer of the CNN by the filter responses to that image. A layer with N_1 distinct filters has N_1 feature maps each of size M_l, where M_l is the height times the width of the feature map. Hence the results in a layer l can be stored in a matrix $F^l \epsilon R^{N_t x M_t}$ where F^l_{ij} is the activation of the i^{th} filter at position j in layer l. We performed gradient descent on white noise image to find any other image that matched the feature responses of the original image. This was done to visualize image information that is encoded at different layers of the network. Let $\vec{p}$ and $\vec{x}$ be the original image and the image that wass generated. Let P^l and F^l their respective feature representation in layer l.

Figure 10: Effect of matching content representation in different layers of CNN

So we defined the squared-error loss between the 2 feature representations as:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2 \tag{1}$$

The derivative of this loss function with respect to the activations in layer l equals:

$$\frac{d\mathcal{L}_{content}}{dF^l ij} = \begin{cases} (F^l - P^l)_{ij} & if F^l ij > 0 \\ 0 & if F^l ij < 0 \end{cases} \tag{2}$$

Using the above equation we computed the gradient with respect to the image $\vec{x}$ using standard error back-propagation. The idea was to keep changing the initially random image $\vec{x}$ until it generated the same response in a certain layer of the CNN as the original image $\vec{p}$.

4.3.2 Style representation

We used a feature space that can be built on top of the filter responses of any network to obtain a representation of the style of the input image. This feature space consisted of the correlations between the filter responses which were given by the Gram matrix $G^l \epsilon R^{N_l x N_l}$ where G^l_{ij} is the inner product between feature maps i and j in the layer l:

$$G^l_{ij} = \sum_k F^l_{ik} F^l jk \tag{3}$$

We got a stationary, multi-scale representations of the input image by including the feature correlations of multiple layers. We did the same thing then as we did in content representation. We visualized the information captured by these style feature spaces built on different layers of network by constructing an image that matched the style representation of a given input image. We used gradient descent from a white noise image to minimize the mean squared distance between the entries of the Gram matrices from the original image and the Gram matrices of the image to be generated. Let $\vec{a}$ and $\vec{x}$ be the original image and the image generated and A^l and G^ltheir respective style representations in layer l. The contribution of layer l to the total loss is:

$$E_t = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2 \tag{4}$$

and the total style loss is:

$$L_s tyle(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l \tag{5}$$

where w_l are the weighting factors of the contribution of each layer to the total loss. We computed the derivative of the loss function with respect to activations in layer l by:

$$\frac{dE_l}{dF^l_{ij}} = \begin{cases} frac 1 N_l^2 M_l^2 ((F^l)^T (G^l - A^l))_{ji} & if F^l ij > 0 \\ 0 & if F^l ij < 0 \end{cases} \tag{6}$$

Figure 11: A: Initialization of Gradient Descent. B: Initialized from the content image. C: Initialized from the style image. Four samples of images initialized from different white noise images

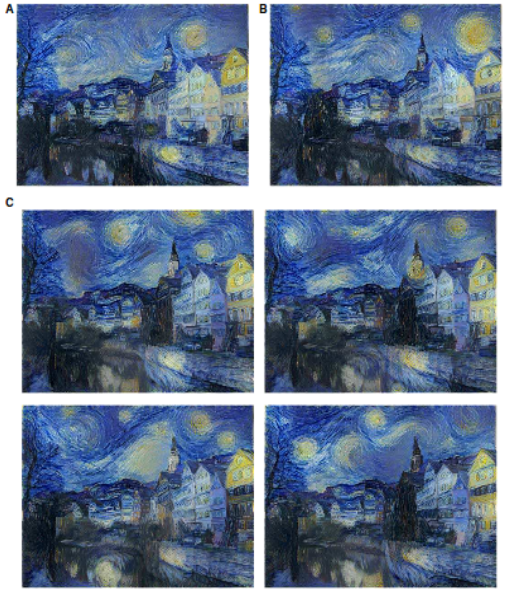


Figure 12: Style Transfer Algorithm

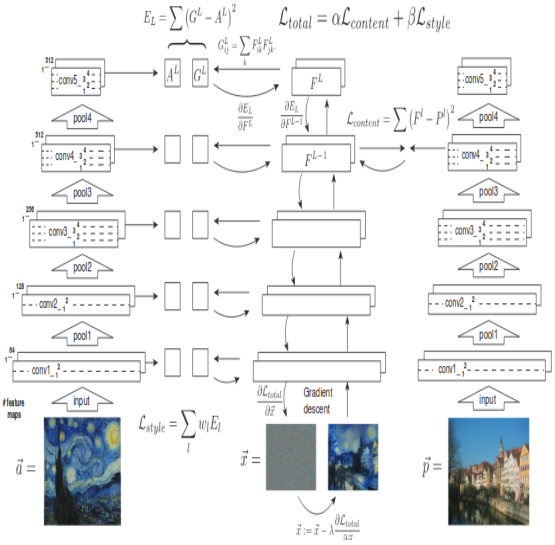


Figure 13: VGG-19 Architecture

4.3.3 Style Transfer

For transferring style of an art work $\vec{a}$ onto a photograph $\vec{p}$ we created a new image that simultaneously matched the content representation of $\vec{p}$ and the style representation of $\vec{a}$. Hence we simultaneously minimized the distance of the feature representations of a white noise image from the content representation of the photograph in one layer and the style representation of the painting defined on a number of layers of the Convolutional Neural Network. Hence the loss function equals

$$L_total(\vec{p}, \vec{a}, \vec{x}) = \alpha L_content(\vec{p}, \vec{x}) + \beta L_style(\vec{a}, \vec{x}) \quad (7)$$

Here α and β are the weighting factors for content and style reconstruction respectively. We performed some optimization by using the gradient with respect to the pixel values $\frac{dL_total}{d\vec{x}}$ as input. Here we tested the performance of 2 optimizers L-BFGS and Adam Optimizer, between which L-BFGS worked better. Before running the model we resized the style image to the same size as the content image.

4.3.4 Architecture

We used a 5 layered CNN- architecture VGG-19. The description of architecture is as following:

```
LAYER GROUP 1
--conv1_1 | shape=(1, 512, 512, 64) | weights_shape=(3, 3, 3, 64)
--relu1_1 | shape=(1, 512, 512, 64) | bias_shape=(64,)
--conv1_2 | shape=(1, 512, 512, 64) | weights_shape=(3, 3, 64, 64)
--relu1_2 | shape=(1, 512, 512, 64) | bias_shape=(64,)
--pool1   | shape=(1, 256, 256, 64)
LAYER GROUP 2
--conv2_1 | shape=(1, 256, 256, 128) | weights_shape=(3, 3, 64, 128)
--relu2_1 | shape=(1, 256, 256, 128) | bias_shape=(128,)
--conv2_2 | shape=(1, 256, 256, 128) | weights_shape=(3, 3, 128, 128)
--relu2_2 | shape=(1, 256, 256, 128) | bias_shape=(128,)
--pool2   | shape=(1, 128, 128, 128)
LAYER GROUP 3
--conv3_1 | shape=(1, 128, 128, 256) | weights_shape=(3, 3, 128, 256)
--relu3_1 | shape=(1, 128, 128, 256) | bias_shape=(256,)
--conv3_2 | shape=(1, 128, 128, 256) | weights_shape=(3, 3, 256, 256)
--relu3_2 | shape=(1, 128, 128, 256) | bias_shape=(256,)
--conv3_3 | shape=(1, 128, 128, 256) | weights_shape=(3, 3, 256, 256)
--relu3_3 | shape=(1, 128, 128, 256) | bias_shape=(256,)
--conv3_4 | shape=(1, 128, 128, 256) | weights_shape=(3, 3, 256, 256)
--relu3_4 | shape=(1, 128, 128, 256) | bias_shape=(256,)
--pool3   | shape=(1, 64, 64, 256)
LAYER GROUP 4
--conv4_1 | shape=(1, 64, 64, 512) | weights_shape=(3, 3, 256, 512)
--relu4_1 | shape=(1, 64, 64, 512) | bias_shape=(512,)
--conv4_2 | shape=(1, 64, 64, 512) | weights_shape=(3, 3, 512, 512)
--relu4_2 | shape=(1, 64, 64, 512) | bias_shape=(512,)
--conv4_3 | shape=(1, 64, 64, 512) | weights_shape=(3, 3, 512, 512)
--relu4_3 | shape=(1, 64, 64, 512) | bias_shape=(512,)
--conv4_4 | shape=(1, 64, 64, 512) | weights_shape=(3, 3, 512, 512)
--relu4_4 | shape=(1, 64, 64, 512) | bias_shape=(512,)
--pool4   | shape=(1, 32, 32, 512)
LAYER GROUP 5
--conv5_1 | shape=(1, 32, 32, 512) | weights_shape=(3, 3, 512, 512)
--relu5_1 | shape=(1, 32, 32, 512) | bias_shape=(512,)
--conv5_2 | shape=(1, 32, 32, 512) | weights_shape=(3, 3, 512, 512)
--relu5_2 | shape=(1, 32, 32, 512) | bias_shape=(512,)
--conv5_3 | shape=(1, 32, 32, 512) | weights_shape=(3, 3, 512, 512)
--relu5_3 | shape=(1, 32, 32, 512) | bias_shape=(512,)
--conv5_4 | shape=(1, 32, 32, 512) | weights_shape=(3, 3, 512, 512)
--relu5_4 | shape=(1, 32, 32, 512) | bias_shape=(512,)
--pool5   | shape=(1, 16, 16, 512)
```

4.3.5 Training

We perform style transfer on various input imges using the following images for extracting style/texture from:

We compared the performances of 2 optimizers, LBFGS and Adam optimizer on the same architecture. Both the op-

Figure 14: Styles used (left to right) teresa, bnw, faces, depp

timizers were ran for a total of 1000 iterations with a learning rate = 1.0. We also replaced Pooling by Average Pool layer and obtained much clearer and better results.

4.3.6 Evaluation metrics

We used the values of α =0.001 and β =0.001 for calculating our total loss. Since we are changing the style and the content of the image, evaluating similarity between the input and the output images here would not have given any insight. So we just measure the values of the total loss for each epoch our optimizers (Adam and LBFGS) are run for. Figures 23 and 24 show the plots of the loss values vs number of epochs for both the optimizers. We see that Adam performs faster than LBFGS though is not able to converge as much. As a result an output image is obtained faster though there is not much transferring of style obtained. LBFGS however works much better and produces clear artistic styled images in slightly more time.

5. Results & Discussion

From the results of the initial task of automatically curating we noticed that finer details like shape of mouth, nose lines, et. were also filtered out while thresholding. During shading the sketches via GAN, we observed that the network can very well learn shadows and performs the corresponding exercise of highlighting using various shades of the same color. Even without dispensing color hint to the network line and shadows showed up really well, though the network expects color hint to perform colored shading. Finer details such as eye color, lip shade, etc. were not preserved and also the coloring goes beyond bounds or leave blank spots as well.

Apart from visually comparing the generated (shaded) image and the original image for evaluating realistic image synthesis models, we also used four different metrics to assess the image similarity and quantify the the evaluation. Figure 15 shows the histogram of image count (out of the 100 random test images) versus their pixel similarity, this clearly depicts the fact that the images are visually similar but not on a pixel level. Similarly, the quality of the images synthesized is not very robust, as observed from Figure 16 which depicts the structural similarity index (SSIM) measuring the quality difference of the generated (shaded) image and the original image.

Figure 15: Pixel Similarity: Image Count vs Value

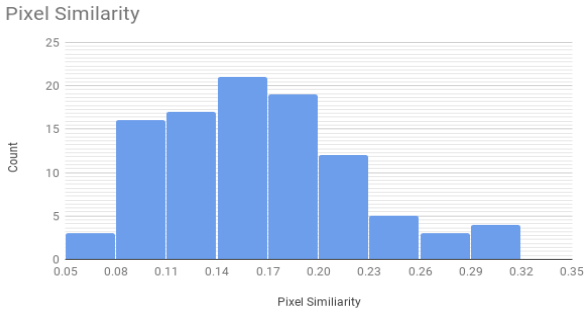
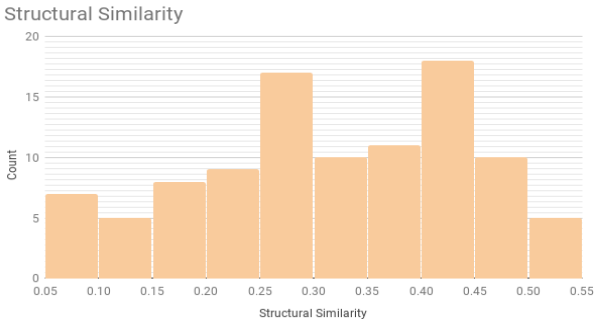


Figure 16: Structural Similarity: Image Count vs Index



When computing the distances between the color histograms of the generated (shaded) image and the original image, we find that the cost *a.k.a.* the Earth Movers Distance ranges between 0.001 to 0.006 as can be seen from Figure 17. On the same line, we find the SIFT similarity to lie mostly between 70% to 90% (refer Figure 18) thus evaluating the success of our model. Figure 19 also shows the euclidean difference between original and shaded images (in $\%$) versus the count of images (out of the 100 tested ones). We infer that difference ranges from 4.62% to 59.19% with the maximum number of images, *i.e.,* 29 images lying in the interval of $6\% - 12\%$ and just 2 images in $42\% - 48\%$.

For the style transfer of the input style image to the obtained shaded image of the model, the results obtained can be seen in the Figure 20 and Figure 21. The loss obtained per epoch for LBFGS and Adam optimizer respectively are shown in Figure 23 and Figure 22. Overall, we conclude that LBFGS works quite better than Adam optimizer though

Figure 17: Wasserstein Metric (Earth Movers Distance)

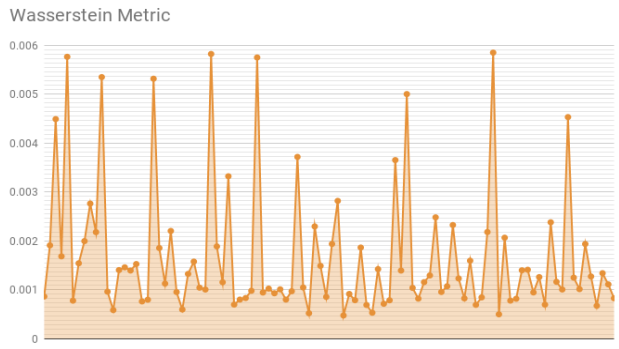


Figure 18: SIFT (Scale-invariant Feature Transform) Similarity Metric: Image Count vs Index

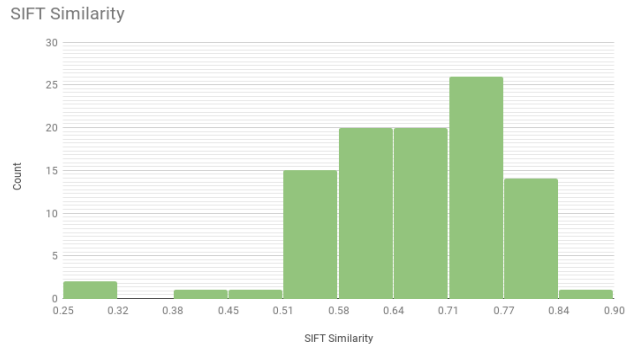
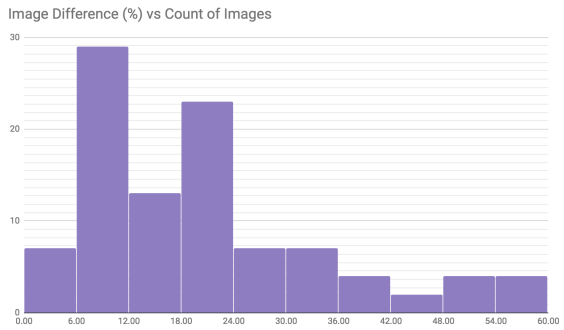


Figure 19: Difference between original & shaded images vs Count



it does take quite significant time (average = 502.6 s) as compared to Adam optimizer's 85.5 s.

Figure 20: LBFGS: Outputs from styles (l-r) teresa, bnw, faces, depp

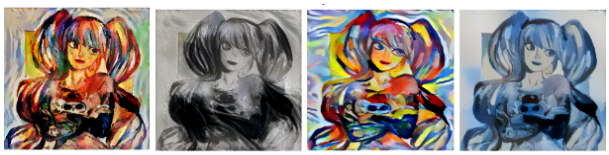


Figure 21: Adam: Outputs from styles (l-r) teresa, bnw, faces, depp

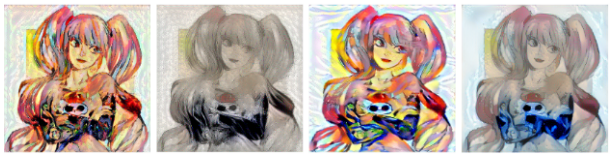
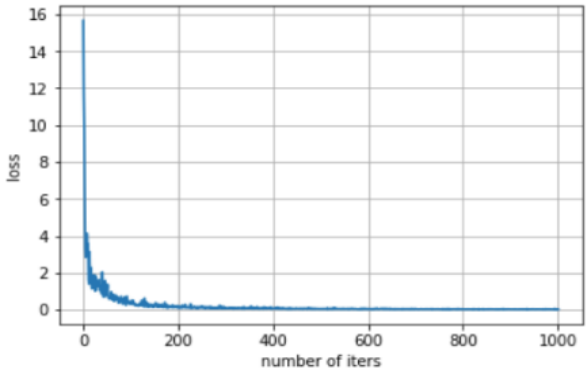


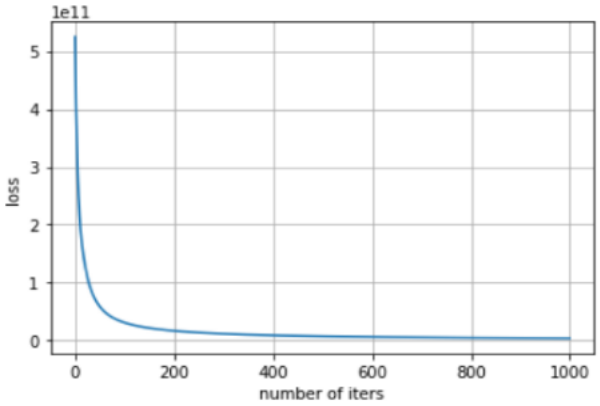
Figure 22: Loss vs Number of iterations for LBFGS Optimizer



6. Conclusion & Future Work

In this project, we worked on fabricating an artificial system based on Deep Neural Networks to curate line art, shade the sketch and further imitate artistic style. Our approach was based on using GAN with a U-Net structure to encode & decode image features for the former part of the image synthesis problem which allowed the shaded image to have both low-level information of sketch as well as learned high-level color information. By introducing residual connections and re-working our way via the leaky ReLU activation function, we were able to synthesize shaded images with a maximum of 90% (SIFT) similarity. In the latter case of style transfer we used the pre-trained VGG-19 model, where the application of LBFGS optimizer ended up giving better results in aesthetics and time as compared

Figure 23: Loss vs Number of iterations for Adam Optimizer



to Adam optimizer.

Despite promising results, our current system suffers from the difficulties of adjusting parameters just like other deep learning models. We also had a very slow training time owing to the complexity of the neural model, *for example* it took around 24 hours for the GAN to train on 15,000 *Manga* images for just 25 epochs on a *GeForce GTX 1080 Ti* GPU.

In the future, we aim to integrate the two neural models to work as a single entity and also extend the same feature to videos, and thus achieve coloured versions of black and white video clips. Improving the performance of the model as well the speed in order to build an interactive toolkit for users also remains and is worth exploring.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *ACM Trans. Graph.*, vol. 28, pp. 24:1–24:11, July 2009.

[2] J. Hays and A. A. Efros, "Scene completion using millions of photographs," *ACM Trans. Graph.*, vol. 26, July 2007.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *ArXiv e-prints*, June 2014.

[4] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015.

[5] E. L. Denton, S. Chintala, A. Szlam, and R. Fergus, "Deep generative image models using a laplacian pyramid of adversarial networks," *CoRR*, vol. abs/1506.05751, 2015.

[6] J. Johnson, A. Alahi, and F. Li, "Perceptual losses for real-time style transfer and super-resolution," *CoRR*, vol. abs/1603.08155, 2016.

[7] H. Dong, P. Neekhara, C. Wu, and Y. Guo, "Unsupervised image-to-image translation with generative adversarial networks," *CoRR*, vol. abs/1701.02676, 2017.

[8] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," *CoRR*, vol. abs/1603.08511, 2016.

[9] P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays, "Scribbler: Controlling deep image synthesis with sketch and color," *CoRR*, vol. abs/1612.00835, 2016.

[10] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," *Proceedings of SIGGRAPH 2001*, pp. 341–346, August 2001.

[11] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, (New York, NY, USA), pp. 327–340, ACM, 2001.

[12] M. Ashikhmin, "Fast texture transfer," *IEEE Comput. Graph. Appl.*, vol. 23, pp. 38–43, July 2003.

[13] H. Lee, S. Seo, S. Ryoo, and K. Yoon, "Directional texture transfer," in *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '10, (New York, NY, USA), pp. 43–48, ACM, 2010.

[14] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, vol. abs/1508.06576, 2015.

[15] Safebooru, "Safebooru - anime picture search engine !," 2017.

[16] H. Wei, Y. Zhao, and J. Ke, "Automatic manga colorization with hint," *CS231n Convolutional Neural Networks for Visual Recognition*, 2017.

[17] Y. Liu, Z. Qin, Z. Luo, and H. Wang, "Auto-painter: Cartoon image generation from sketch by using conditional generative adversarial networks," *CoRR*, vol. abs/1705.01908, 2017.

[18] K. Frans, "Deepcolor: Automatic coloring and shading of manga-style line-art," 2017. http://kvfrans.com/coloring-and-shading-line-art-automatically-through-conditional-gans/.

[19] L. Zhang, Y. Ji, and X. Lin, "Style transfer for anime sketches with enhanced residual u-net and auxiliary classifier GAN," *CoRR*, vol. abs/1706.03319, 2017.

[20] S. Shi and X. Chu, "Speeding up convolutional neural networks by exploiting the sparsity of rectifier units," *CoRR*, vol. abs/1704.07724, 2017.

[21] J. Miguel, "Activation functions in deep learning," 2017.

[22] K. Frans, "Outline colorization through tandem adversarial networks," *CoRR*, vol. abs/1704.08834, 2017.

[23] K. G. Larkin, "Structural similarity index ssimplified: Is there really a simpler concept at the heart of image quality measurement?," *CoRR*, vol. abs/1503.06680, 2015.

[24] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *Int. J. Comput. Vision*, vol. 40, pp. 99–121, Nov. 2000.

[25] J. Li and G. Wang, "An improved sift matching algorithm based on geometric similarity," in *2015 IEEE 5th International Conference on Electronics Information and Emergency Communication*, pp. 16–19, May 2015.