Style Brush: Interactive Style Transfer

Kunal Chhabria
Department of Computer Science
Simon Fraser University
Burnaby, BC
kunal_chhabria@sfu.ca

Priyanka Chandrashekar
Department of Computer Science
Simon Fraser University
Burnaby, BC
priyanka_chandrashekar@sfu.ca

Abstract

We consider a modern application of style transfer and super resolution, both of which are some of the most common image transformation problems. We build an application that allows users to either capture an image, or send in a pre-captured image, and paint over it with eight painting styles as per their choice. Finally, once the user is satisfied with their choices, they can pass it through a super resolution model so as to form a final up-scaled painting that is ready to print. The style transfer is built with inspiration from work by Johnson et al. [1] to receive aesthetically pleasing results. The super resolution is implemented from work by Wang et al. [2] to form a fast working generative adversarial network for super resolution. The final application is tested with a variety of images, both captured and uploaded, and it has worked on all types of images with great speed and produced appealing, custom results.

1 Introduction

Image transformation problems have been a key part of computer vision, the most well-studied of which are style transfer and super resolution. Style transfer allows users to transform their images into visual masterpieces, similar to the work of other paintings, while super resolution transforms low resolution smaller sized images into high resolution up-scaled images. However, both of these processes allow little to no input from the users. The users must make their choices beforehand about the type of painting they want etc.

This is what drove us to build the Style Brush application. It is a complete application built on Python that gives users freedom to choose how to style their images. The idea is to build an application that allows users to style their images as per their wish and finally receive a custom painting to capture the moment instantly, similar to a photo-booth but with more user

Figure 1: A sample image with eight trained styles on it

input. The users can capture an image at the same moment, or instead upload a previously chosen image. The application then runs on eight different style transfer models corresponding to eight painting styles. The eight stylized images are then presented to the user as choices on an interactive user interface, where the user can draw on their image to choose an area to be styled. Once the user makes a selection, a mask is generated for the area chosen, blurred slightly so that the merging between two styles isn't too sharp and then the chosen style is applied. The user then has the option to go back and continue styling other parts of the image with some of the other styles as per their wish. If not, they can exit the process, upon which the final image is passed through the super resolution model, which outputs a final up-scaled high resolution image that is ready for the user to print.

Style transfer is the process of taking style from one image and applying it to another image. Gatys et al. [5] first showed that the activation maps in CNNS can be used to get accurate content and style information and blend them in a meaningful way. But their approach is slow as it modifies the image itself. We studied three approaches to do style transfer in real time namely Johnson et al. [1], Ulyanov et al. [6], Li and Wand [7], and Johnson et al produced most appealing results. Further Ulyanov et al. [8] introduced Instance Normalization which significantly improves image quality. Hence, we combine these approaches to do style transfer in our application.

The scribble user interface is built using Tkinter on Python, to give the user freedom to choose as many styles as they like on varying regions, how many ever times they would prefer. The region coordinates are passed on to build a mask that is blurred using a 25x25 Gaussian blur, so that individual styles merge together smoothly. The mask is then used to implement the style on the previous image to present to the user. This gives a final amalgamation of styles that looks pleasing to the eye.

To build the super resolution module, extensive background work was done by implementing three pioneering and state of the art works in this area; EnhanceNet [9], Super Resolution GAN (SRGAN) [4] and Enhanced Super Resolution GAN (ESRGAN) [2]. However, after comparing the results for quality, detail and speed, finally for the software the implementation of ESRGAN, the work by Wang et al. [2] was chosen. The ESRGAN is a generative adversarial network which was built with inspiration from the work by Ledig et al. [4], with a few deviations, which leads to more detail in the final result. Three changes are done, which can be summarized as a change in the network architecture to remove Batch Normalization layers and replace the blocks by Residual-in-Residual Dense Blocks (RRDBs), to replace the discriminator in the SRGAN with a Relativistic Discriminator [3], and to use perceptual loss to get more realistic images.

2 Working

The entire project can be divided into three modules: style transfer, interactive interface and super resolution.

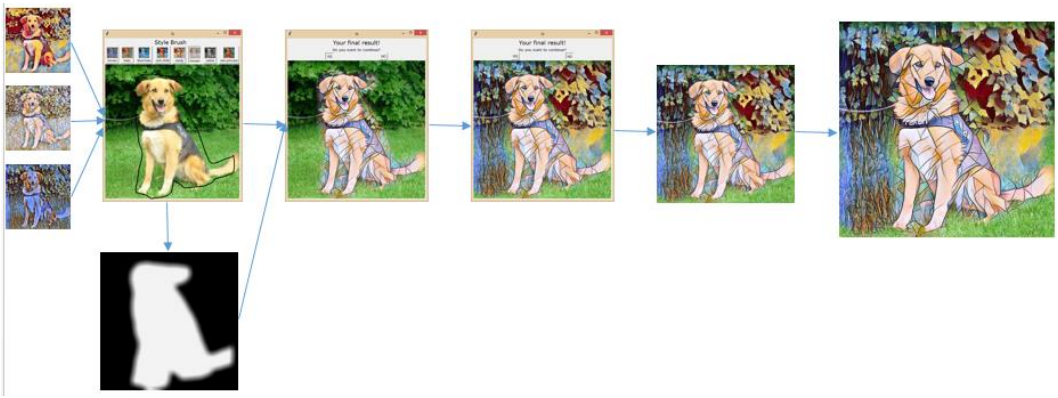

Figure 2: Complete flow of processes

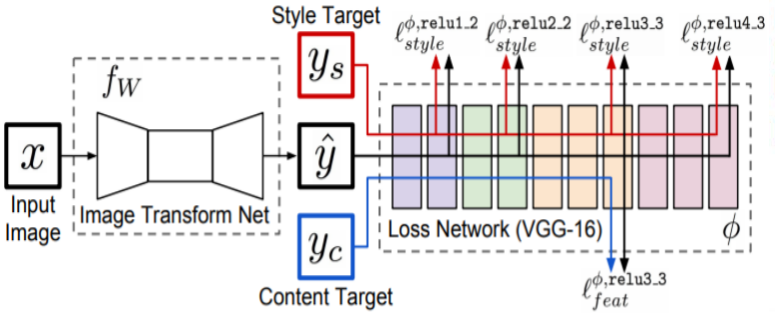

Figure 3: Network Architecture [1]

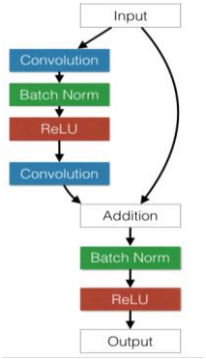

Figure 4: Residual block [1]

2.1 Style Transfer

We use the approach by Johnson et al. to produce multiple styled output images for an input image as it can produce 8 images in less than 5 seconds and the images are visually appealing.

2.1.1 Network Architecture

The architecture contains an Image Transformation Network which is a deep residual Convolutional Neural Network and a loss network which is a VGG-16 model pre-trained on ImageNet dataset (Figure 3). The Image Transformation network transforms input image (x) into output image (y). This network is trained to minimize the feature reconstruction loss and style reconstruction loss obtained from the loss network using stochastic gradient descent as follows:

$$W^* = \arg\min_W E_{x,\{yi\}} \sum_{i=1}^{n} \lambda_i \mathrm{l}_i(f_W(\mathrm{x}), \mathrm{y}_i)$$

Johnson et al. use 5 residual blocks as shown in Figure 3, with a BatchNorm and ReLU between each block. However, we replace Batch Norm with Instance Normalization throughout the Image Transformation Network.

2.1.2 Instance Normalization

Let x ∈ R and x= T×C×W×H be an input tensor where a batch consists of T images. Let x_{tijk} be the tijk-th element, where k and j span spatial dimensions, i be the feature channel/color channel and t be the image index for that batch.

They propose replacing batch normalization with instance normalization given by:

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \varepsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_{ti})^2$$

2.1.3 Feature Reconstruction Loss

Since we don't want the pixels in output image to exactly match the pixels of content image, we want them to have similar feature representations as that of loss network φ . φ_j(x) are the activations in the j_{th} layer of the network φ for image x; j will be a convolutional layer and φ_j(x) will be a feature map of shape $C_j \; x \; H_j \; x \; W_j$. Here, the activations of layer relu2_2 are used.

$$l_{feat}^{\varphi,\mathrm{j}}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \left\| \varphi_j(\hat{y}) - \varphi_j(y) \right\|_2^2$$

2.1.4 Style Reconstruction Loss

We want the output image to have similar colors, patterns, textures as the style image. Gatys et al propose the following style reconstruction loss. Gram matrix for a feature map x of shape $C_j \; x \; H_j \; x \; W_j$, $G_j^{\varphi}(x)$ is a $C_j \; x \; C_j$ matrix whose elements are given by:

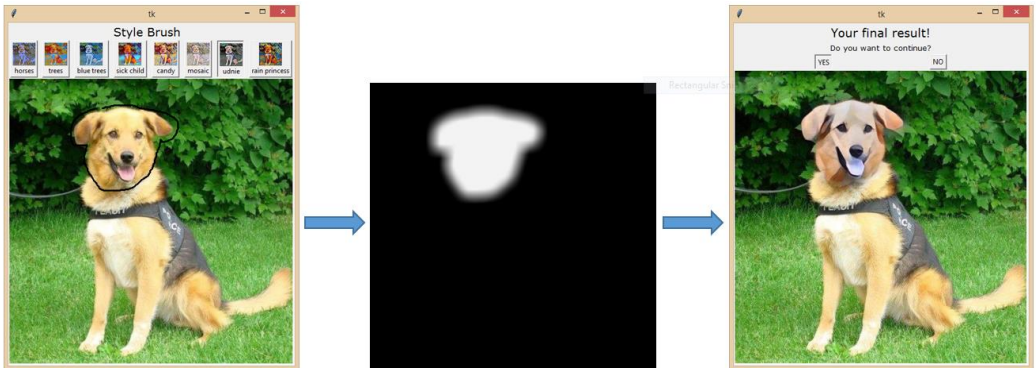

Figure 5: Interactive interface where 1) image is drawn on 2) mask is created and 3) final created image is presented

$$G_j^{\varphi}(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \varphi_j(x)_{h,w,c} \varphi_j(x)_{h,w,c'}$$

The style reconstruction loss is Frobenius norm of difference of Gram matrix of output and target image:

$$l_{style}^{\varphi,j}(\hat{y}, y) = \left\| G_j^{\varphi}(\hat{y}) - G_j^{\varphi}(y) \right\|_F^2$$

We sum the loss for each layer j ∈ J and J= relu1_2, relu2_2, relu3_3, and relu4_3.

2.2 Interactive Interface

The interactive interface is the only way the user can interact with the style transfer models. Built on Tkinter, it is built to show the styled images returned by the eight models, so that the user can look at them in the buttons and decide the styles they would want on their image. After choosing a style by clicking on its button, the user can draw on the image. Tkinter works in the background to collect the coordinates of the region chosen by the user. The coordinates are then passed on the OpenCV to build the mask corresponding to the region. After the mask is built, a Gaussian Blur of 25x25 dimension is applied, so that the edges of the mask get smoothed out slightly.

Once the final mask is ready, all that is left is combine the image and the styled image chosen with the help of the mask (Figure 5 2)). For this, the following equation is used:

$$Img_{Final} = Img_{Source} \times Mask + Img_{Styled\ Source} \times (1 - Mask)$$

The final image with the combined source and styled image is presented to the user, so that they can make a choice whether they want to continue styling the image or leave it at this stage. If they choose to continue the whole process is repeated, if not the image is passed on to the next module.

2.3 Super Resolution

The next step to building our final painting is to upscale the user's final choices in a way that maintains the sharpness of the ground truth image, so that when the user prints out their

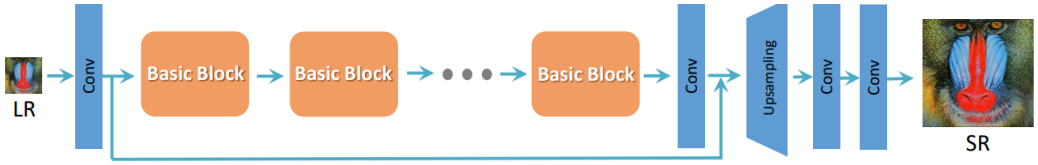

Figure 6: Network Architecture for ESRGAN [2]

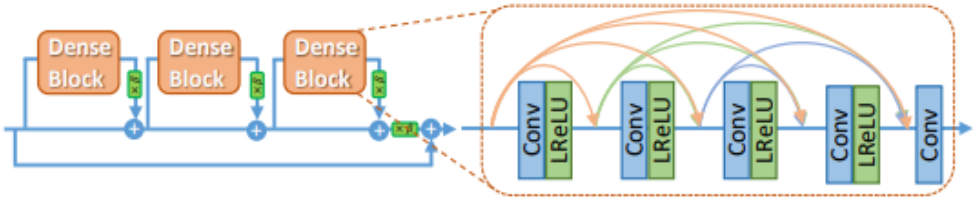

Figure 7: Residual-in-Residual Dense Block (RRDB) [2]

painting its details are preserved. The ESRGAN [2] overcomes a major drawback of SRGAN [4] and EnhanceNet [9] which is lack of high-resolution detail. The following are the major changes made to the SRGAN model in the chosen ESRGAN.

2.3.1 Network Architecture

To improve the quality of the image received from SRGAN, two changes are made. Firstly, all batch normalization layers are removed. This is because most of the lack of detail and unnecessary artifacts in the final reproduced image in SRGAN can be attributed to these batch normalization layers, and they may also lead to a slight instability and lack of generalization while certain images are used during training.

Secondly, after removing the BN layers instead of simply using the base blocks, the internal ReLU and Convolutional layers are repeated internally multiple times and connected in a deeper complicated structure. This architecture is called the Residual-in-Residual Dense Block (Figure 7), which has dense connections that the final architecture can benefit from while maintaining details. Since this model is very deep, to help with training a few other techniques are used, like residual scaling and smaller initialization.

2.3.2 Relativistic Average Discriminator

Instead of using the standard discriminator proposed in SRGAN, Relativistic average Discriminator RaD is used, represented as D_{Ra} here, as proposed in Relativistic GAN [3]. It tries to find the probability that the given real image is more "real" or realistic than the sampled fake image. IT leads to better quality images as compared to a standard discriminator. It can be expressed as $D_{Ra}(x_r, x_f) = \sigma(C(x_r) - \mathrm{E}_{x_f}[C(x_f)])$, where $\mathrm{E}_{x_f}[.]$ represents taking the average of all the fake data in the batch.

Discriminator loss can be expressed as:

$$L_D^{Ra} = -E_{x_r}\left[\log\left(D_{Ra}(x_r, x_f)\right)\right] - E_{x_f}\left[\log\left(1 - D_{Ra}(x_f, x_r)\right)\right]$$

Generator loss can be expressed as:

$$L_G^{Ra} = -E_{x_r}\left[\log\left(1 - D_{Ra}(x_r, x_f)\right)\right] - E_{x_f}\left[\log\left(D_{Ra}(x_f, x_r)\right)\right]$$

Here, $x_f = G(x_i)$ and x_i stands for the input image.

2.3.3 Perceptual Loss

Borrowing from the model used for style transfer, Perceptual loss [1] is used here in the generator, but instead of using it on the activation layers, here in ESRGAN it is used before the activation layers to avoid the sparsity in the activation layers and to maintain brightness of the original input image. We can express final generator loss as:

$$L_G = L_{percep} + \lambda L_G^{Ra} + \eta L_1$$

where $L_1 = \mathrm{E}||G(x_i) - y||_1$ is the content loss evaluating 1 norm distance between the original image and the recreated image.

Figure 8: Applying 8 styles to an image

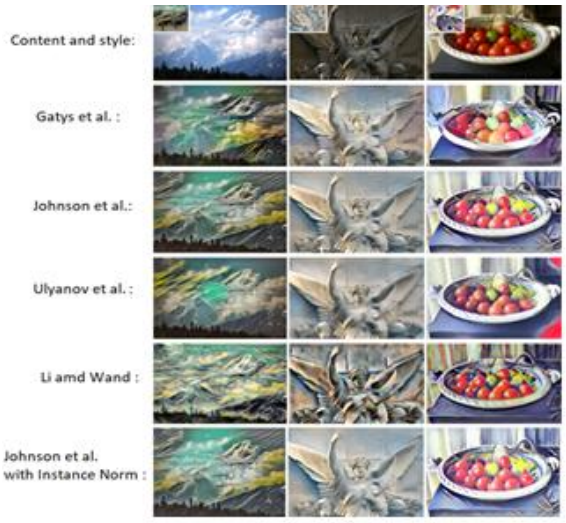

Figure 9: Comparison with previous works [10]

3 Experiments

3.1 Style Transfer

The software requires 8 different styles, hence for each style, a new model must be trained. We used Microsoft COCO 2014 [11] dataset which has 80,000 images. We trained 8 different models and training takes 3.5 hours for 2 epochs on a single Nvidia RTX 2060 GPU (Figure 8 and 9).

3.2 Super Resolution

Choosing a final model consisted of implementing three works of super resolution: EnhanceNet, SRGAN and ESRGAN. However, we finally chose our implementation of ESRGAN due to the high level of detail achieved in the final result, which is desirable to get photo realistic image as we can see in the close up comparison between original model results and those of our model. The training time for the ESRGAN model was around 26 hours, as it was trained over 200 epochs. The training data consists of DIV2K (800 images) [12], Flickr2K (2650 images) [13] and OutdoorSceneTraining (OST) (10,324 images) [14] (Figure 10 and 11).

4 Conclusion

After comprehensive background work, we can conclude that style transfer built on perceptual loss with instance normalization and super resolution built as ESRGAN are the best choices for the interactive style transfer brush application we built. They produced great results at a good speed for an application, which led to a well-integrated complete application.

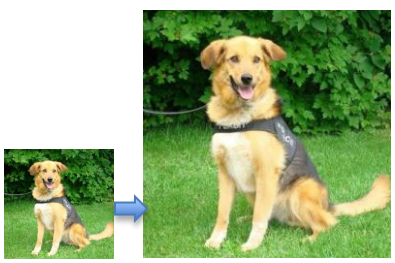

Figure 10: After super resolution

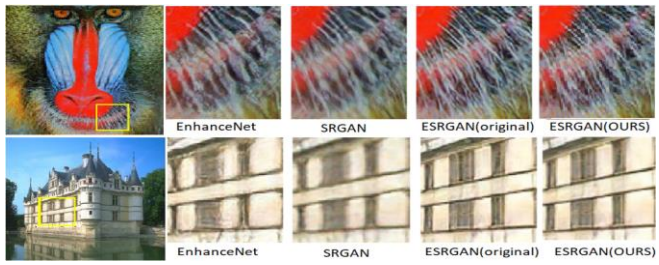

Figure 11: Comparison with previous works [2]

5 Contributions

The building of the application was a long and complex task. For this, the background and implementation corresponding to the style transfer module were handled by Kunal Chhabria, while the background and implementation for super resolution were handled by Priyanka Chandrashekar. This included researching potential papers and models, implementing them, training them, comparing their results, choosing one that works best with the application and integrating it with the application.

The rest of the interactive interface was built by both Kunal and Priyanka together, which included brainstorming ideas to form the interface, studying up on Tkinter, integrating various moving components of the interface together and after everything is complete testing that the application is functional.

References

[1] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." *European conference on computer vision*. Springer, Cham, 2016.

[2] Wang, Xintao, et al. "Esrgan: Enhanced super-resolution generative adversarial networks." *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[3] Jolicoeur-Martineau, Alexia. "The relativistic discriminator: a key element missing from standard GAN." *arXiv preprint arXiv:1807.00734* (2018).

[4] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017.

[5] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "A neural algorithm of artistic style." *arXiv preprint arXiv:1508.06576* (2015).

[6] Ulyanov, Dmitry, et al. "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images." ICML. Vol. 1. No. 2. 2016.

[7] Li, Chuan, and Michael Wand. "Precomputed real-time texture synthesis with markovian generative adversarial networks." *European Conference on Computer Vision.* Springer, Cham, 2016.

[8] Ulyanov, Dmitry, Andrea Vedaldi, and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization." arXiv preprint arXiv:1607.08022 (2016).

[9] Sajjadi, Mehdi SM, Bernhard Scholkopf, and Michael Hirsch. "Enhancenet: Single image super-resolution through automated texture synthesis." *Proceedings of the IEEE International Conference on Computer Vision.* 2017.

[10] Jing, Yongcheng, et al. "Neural style transfer: A review." *IEEE transactions on visualization and computer graphics* (2019).

[11] Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." *European conference on computer vision.* Springer, Cham, 2014.

[12] Agustsson, Eirikur, and Radu Timofte. "Ntire 2017 challenge on single image super-resolution: Dataset and study." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017.

[13] Timofte, Radu, et al. "Ntire 2017 challenge on single image super-resolution: Methods and results." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.* 2017.

[14] Wang, Xintao, et al. "Recovering realistic texture in image super-resolution by deep spatial feature transform." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018.