Style Transfer

Sahiti Priya Chittam
schittam@ucsd.edu

Richa Pallavi
rpallavi@ucsd.edu

Mahima Rathore
mrathore@ucsd.edu

Sushmitha Rao Uppugunduri
suppugun@ucsd.edu

Abstract

This project explores methods for artistic style transfer based on convolutional neural networks. The core idea proposed by Gatys et. al (1) became very popular and with further research Johnson et. al (2) overcame a major limitation to achieve style transfer in real-time. We implement both the approaches in this project. For both conventional and real-time style transfer, we use style and content images from wikiart and flickr_landscape respectively. For real-time style transfer we use Microsoft's COCO dataset to train an image transformation network for every desired style. We observe that, while both approaches produce transformed images that are largely similar in quality, real-time style transfer offers a major benefit, in that, once a style has been trained, it can be applied on a large set of content images with almost no overhead. Therefore we apply a few styles on a variety of images and analyse the qualitative features of the content image that generates an aesthetically pleasing style transferred image.

1 Introduction

Style transfer is the technique of synthesizing artistic images by combining the content of one image with the style of another. Convolutional Neural Networks are best suited for this task because they learn a hierarchy of feature representations. Recent methods for style transfer on images fall into two categories – optimization-based methods and feed-forward methods. Optimization-based approaches such as the one proposed in Gatys et. al (1) solve an optimization problem for each synthesized image. They give high-quality results but can take minutes to synthesize each image. Feed-forward methods as the one proposed by Johnson et. al (2) train neural networks to approximate solutions to these optimization problems so that after training, they can be applied in real-time.

1.1 Motivation

Style transfer facilitates the transformation of daily pictures to artistic style images. This algorithm can be utilized to create fancy camera filters similar to Prisma and Facebook Live Video. The styling of images can also be leveraged in computer graphics tasks such as painting the environments for computer games, virtual reality games, and animation clips where the style can be applied as a texture on the rendered content. When paired with object detection, it could be valuable in interior and fabric design domains by selectively applying styles, potentially enhancing their respective e-commerce operations; the application could help the customer visualize his/her choice of style/design on a product. For example, a wardrobe remodelling interior design task could provide various styles of decorative laminated sheets to choose from and display the stylized wardrobe when the customer uploads a picture of his/her wardrobe and surroundings. This project demonstrates how machine learning techniques can achieve considerable success in the field of art and creativity.

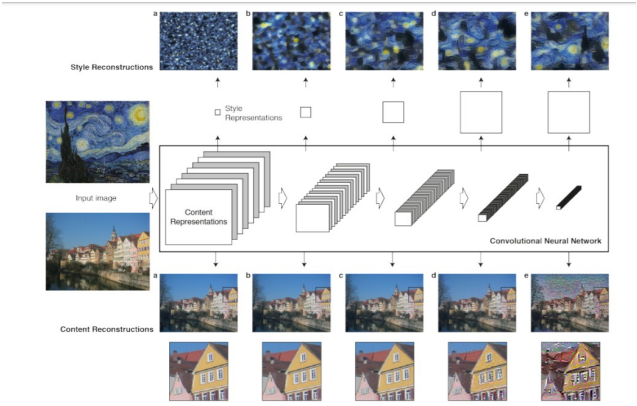

Figure 1: Neural Style Transfer Architecture

2 Description and Methods

2.1 Neural Style Transfer

Neural style transfer deploys an optimization technique which takes two images — a content image and a style reference image (such as an artwork by a famous painter), and blends them together to obtain an output image which looks like the content image, but "painted" in the style of the style reference image. This is implemented by optimizing the output image to match the content statistics of the content image and the style statistics of the style reference image. These statistics are extracted from the images using a convolutional network.

2.1.1 Algorithm

The principle of neural style transfer is to define two distance functions, one that describes how different the content of both images are, $L_{content}$, and one that describes the difference between the two images in terms of their style, L_{style}. Then, given three images, a desired style image, a desired content image, and the input image, the input image is transformed to minimize the content distance with the content image and its style distance with the style image.

2.1.2 Architecture

Intermediate layers are used to represent feature maps that become increasingly higher ordered as you go deeper. We are using the network architecture VGG19, a pretrained image classification network. These intermediate layers define the representation of content and style from our images. For an input image, we will try to match the corresponding style and content target representations at these intermediate layers. Hence by accessing intermediate layers, we're able to describe the content and style of input images. As a result, somewhere between where the raw image is fed in and the classification label is output, the model serves as a complex feature extractor.

2.1.3 Equations

Given the original image p and the generated image x, we define P^l and F^l as the feature response in layer l, reshaped as a 2D matrix for the original image and the generated image respectively. The content loss is then given by:

$$L_{content} = \frac{1}{2} \sum_{i,j} (F^l_{i,j} - P^l_{i,j})^2 \tag{1}$$

For style-loss we need to compute feature correlation given by the Gram matrix, $G^l_{i,j}$ where $G^l_{i,j}$ is the inner product between the vectorized feature map i and j in layer l:

$$G^l_{i,j} = \sum_k F^l_{i,k} F^l_{j,k} \tag{2}$$

To generate a texture that matches the style of a given image we minimize the mean-squared distance between the entries of the Gram matrix from the original image and the Gram matrix of the image to be generated. So let a and x be the original image and the generated image and A^l and G_l their respective style representations in layer l. The contribution of that layer to the total loss is then

$$\frac{1}{4N^2M_l^2}\sum_{i,j}(G_{i,j}^l - A_{i,j}^l)^2 \tag{3}$$

and the total loss is given by

$$L_{style}(a,x) = \sum_{l=0}^{L} w_l E_L \tag{4}$$

The final loss function that we need to minimize is (p is the content image and a is the style image):

$$L_{total}(p,a,x) = \alpha L_{content} + \beta L_{style} \tag{5}$$

2.2 Real Time Style Transfer

Neural style transfer generates high-quality images by optimizing perceptual loss functions based on high-level features extracted from a pretrained network. But it cannot be applied in real-time because it involves solving an optimization problem(1). Other methods for such problems typically train feed-forward CNNs using a per-pixel loss between the output and ground-truth images(3). Johnson et. al combined the advantages of both approaches to perform feedforward style transfer while matching the high-quality results produced by the optimization-based methods(2). The same is implemented in this project.

2.2.1 Algorithm

A feed forward transformation network is used for image transformation. This is trained using perceptual loss functions that depend on high-level features extracted from a pretrained loss network. There are two loss functions calculated: Feature reconstruction loss(l_{feat}) and Style reconstruction loss(l_{style}). Feature reconstruction loss is the difference in feature representations at a particular layer of the loss network for the transformed image and the target content image. Style reconstruction loss is the difference in style, calculated by the gram matrix, between the transformed image and the target style image. A weighted combination of the feature and style losses are minimized during training time. Each specific style must have its own neural network trained to apply real-time style transfer. After completion of training, the style network is equipped to take any input image and forward propagate to generate a stylized version of the image.

2.2.2 Architecture

The model implemented for Image Transformation Network is a deep residual CNN parameterized by weights W. As defined by (2), it contains 3 convolutional layers, 5 residual blocks, and 3 deconvolutional layers. It transforms input images x into output images y' via the mapping y' = $f_W(x)$.

The fact that CNNs pretrained for image classification have already learned to encode the necessary perceptual information is used to derive the loss functions from a fixed pretrained Loss Network. In this project VGG-16 is used. Loss functions defined at different layers of this network compute a scalar value $l_i(y_i, y')$ measuring the difference between the output image y' and a target image y_i. The image transformation network is trained using stochastic gradient descent to minimize a weighted combination of these loss functions. Specifically, the aforementioned Feature reconstruction loss and Style reconstruction loss are used in this project. Figure 2 illustrates the overall architecture for Real time style transfer.

2.2.3 Equations

Feature Reconstruction Loss is given by:

$$l_{\text{feat}}{}^{(L,j)}(y',y) = \frac{1}{C_jH_jW_j}||L_j(y') - L_j(y)||_2^2 \tag{6}$$

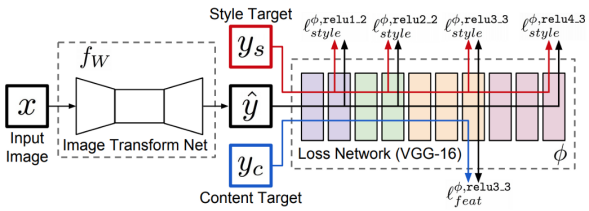

Figure 2: Real time Style Transfer Architecture

where $L_j(x)$ is the activation of the j^{th} layer of the network while processing the image x. If j is a convolutional layer then $L_j(x)$ will be a feature map of shape C_j x H_j x W_j.

Style Reconstruction Loss is given by:

$$l_{\text{style}}{}^{(y',y)} = ||G_j^L(y') - G_j^L(y)||_F^2 \tag{7}$$

where G is the same Gram Matrix as defined in Neural Style Transfer.

3 Implementation Details

3.1 Neural Style Transfer

To implement Neural Style Transfer we define content and style representations from the data-sets provided. We define and load VGG19 model, and feed in our input tensor to the model. This will allow us to extract the feature maps (and subsequently the content and style representations) of the content, style, and generated images.

We have to now define and create our loss functions for content and style distances. For content loss, we pass both the desired content image and our base input image to the network. This will return the intermediate layer outputs from our model. Then we simply take the mean squared loss for our the input. We perform backpropagation in the usual way such that we minimize this content loss. We thus change the initial image until it generates a similar response in a certain layer as the original content image.

Style loss of the base input image, x, and the style image, a, is defined as the distance between the style representation (the gram matrices) of these images. We describe the style representation of an image as the correlation between different filter responses given by the Gram matrix G^l, where $G^l_{i,j}$ is the inner product between the vectorized feature map i and j in layer l. To generate a style for our base input image, we perform gradient descent from the content image to transform it into an image that matches the style representation of the original image. We do so by minimizing the mean squared distance between the feature correlation map of the style image and the input image.

This is followed by running Gradient Descent and computing the loss and gradients. We define a function that loads our content and style images and feed them forward through our network. This outputs the content and style feature representations from our model. Finally we apply and run the style transfer process to complete the transfer on the input images.

3.2 Real Time Style Transfer

Given the architecture for Real-time style transfer which consists of an image transformation network and the VGG-16 pretrained loss network, we train the image transformation network to reduce the overall style transfer loss. For training, we optimize loss of a desired style image over Microsoft COCO dataset that contains around 80000 training images. The process involves using the pretrained VGG network to evaluate the overall loss, which is a weighted combination of the Feature reconstruction loss and Style reconstruction loss for a given training example, and then propagate this error back through every layer in the image transformation network. i.e performing stochastic gradient descent on the image transformation network. We then apply the desired style transformation network on the content images to transform images.

4 Experimental Setting

4.1 Data Set

4.1.1 Neural Style Transfer

Data sets are picked from path mentioned below:

- Style images: /datasets/ee285f-public/wikiart/wikiart/
- Content images: /datasets/ee285f-public/flickr_landscape/

4.1.2 Real-Time Style Transfer

The image transformation network in Real-time style transfer is trained on Microsoft COCO dataset. We trained one network for each of 4 style targets using all 80000 training images. The style and content images were picked from the same wikiart and flickr_landscape links as in Neural style transfer. The style images can be found here - Figure 8. We have tested the transform on several content images and have not included the images to conserve space in the document.

4.2 Training Parameters

Both Neural Style Transfer and Real-time style transfer employ hyper-parameters to tune the model to achieve desired balance of content and style in the transferred image:

	Style weight	Content weight
Neural style transfer	1000000	1
Real-time style transfer	10e10	10e5

Table 1: Weights obtained after fine-tuning

4.3 Hardware Used

UCSD's DSMLP Instructional GPU cluster, a service of ITS/Educational Technology Services (formerly ACMS), provides students in all disciplines/divisions access to 80+ modern GPUs running on 10 physical hardware nodes located at SDSC.

DSMLP jobs are executed in form of docker "containers" using KuberNetes container managemen-t/orchestration system.

The resources available for training were: CPU Cores: 8 (Intel Xeon) CPU RAM: 64 GB GPU: NVIDIA GTX 1080Ti GPU RAM: 11GB Network: Arista 7150 10GB Ethernet Switch

5 Results

5.1 Figures

5.1.1 Neural Style Transfer

Figure 3 and Figure 4 corresponds to the content and style images selected from data sets. Figure 5 shows the successful transfer of 2 content images over 3 style images.

For different alpha and beta values Figure 11 shows the successful transferred images of content on style image Figure 6.

5.1.2 Real-Time Style Transfer

Figure 8 row 1, corresponds to the style images used for training real-time style transfer networks and rows 2 and 3 correspond to style transferred images. Figure 10 shows the result with different content and style weights.

Figure 3

Figure 4

```
display_demo_result(output, content_imgs)
```

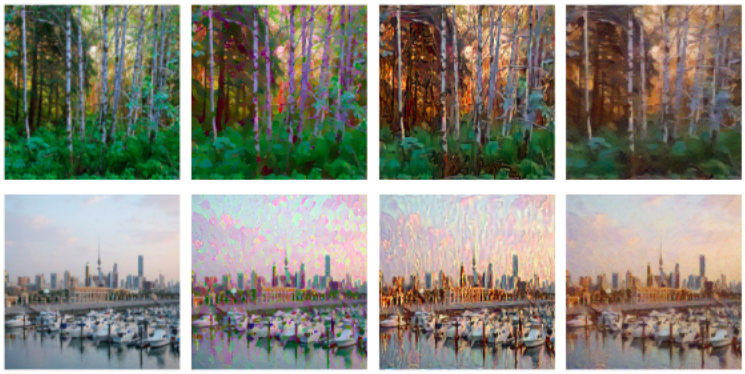

Figure 5: Success Case of style transfer

Figure 6: Neural style transfer - style and content Image

5.2 Comparisons

The Real time style transfer approximates the solution to the optimization problem based on loss minimized on the training set. In general, we observed that the transformed images from real time style transfer are largely similar to the ones obtained in Neural style transfer. Real-time style transfer provides an added benefit that once the model is trained on the training set, it can transform several content images in real-time though a simple feed-forward pass, where the neural style transfer lags.

(a) alpha=1,beta=1000 (b) alpha=1,beta=10000 (c) alpha=1,beta=1000000

Figure 7: Performance with different style and content weight

Figure 8: Comparison of style transfer on different content images; row 1: style image; row 2 and 3 : transformed images

5.3 Plots

Content, style loss for neural-style transfer and perceptual loss for real-time style transfer is plotted for Monet style in Figure 12.

5.4 Success Cases

Figure 9 shows two cases of successful neural style transferred outputs.

With real-time style transfer, Claude Monet style gave best results with transformed images that appear like water color illustrations. The weights for monet network are 10e5 for content and 10e11 for style, trained over 2 epoch.

Figure 9: Success case of real-time style transfer - monet style; row 1: content images, row 2: respective stylized images

(a) Original (b) style weight = 10e11 (c) style weight = 10e10

Figure 10: Performance with different style and content weight

5.5 Failure Cases

We did not observe any case without style transformation using neural style transfer. All the experimental cases did produce a style transferred image. In real-time style transfer, we observed unappealing grainy images with Vincent Van Gogh's Irises for style weight = 10e11 when run over 2 epoch, refer Figure 10.

(a) alpha=1,beta=1000 (b) alpha=1,beta=10000 (c) Real-time style transfer

Figure 11: Performance of Neural Style transfer with different style and content weights against real-time style transformed image

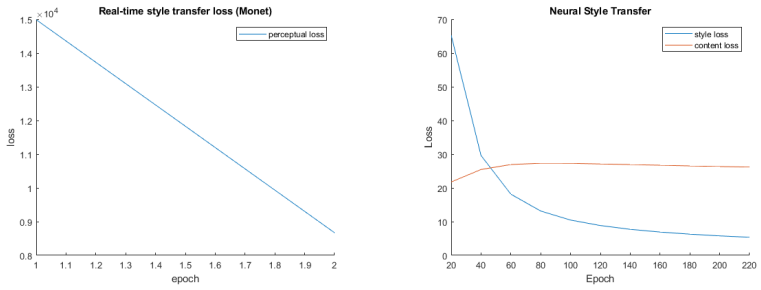

Figure 12: Loss for Monet style

6 Discussion

6.1 Inference

We observed satisfactory perceptual quality on a number of content and style images that we used for style transfer. By tweaking the content and style weights, we were able to modify the flavour of the resultant image. We also observed that, in some cases, the synthesised images are subject to low-level noise. While this is less of an issue in the artistic style transfer, the problem becomes more apparent when the content or the style image or both have a lot of details, i.e. when the style image resembles a photograph and the photo-realism of the synthesised image is affected. This can be attributed to the evaluation criterion which are are not universally applicable on different categories of styles or content images. Besides, this is not a major concern since it is possible to construct denoising techniques to post-process the transformed images.

6.2 Conclusion

In this project, we implemented Neural style transfer and Real-time style transfer and observed its effect on a variety of content and style images. In both methods, we varied the content and style weights and observed the visual changes in the resultant stylized image. Even though, real time style transfer approximates the solution to the optimization problem based on loss minimized on the training set, we observed that the transformed images are largely similar to the ones obtained in Neural style transfer. Therefore real-time style transfer provides an added benefit that once the model is trained on the training set, it can transform several content images in real-time though a simple feed-forward pass.

6.3 Learning

Learning and understanding how the architecture of Neural and Real-Time Style Transfer works was one of the most rewarding outcomes of this project. We were exposed to capabilities and applications of the architecture as a whole. In addition to this, major takeaway was our better understanding of the PyTorch framework as well as packages like numpy,etc. We also gained familiarity with version control systems like Git, and were able to improve our Python programming practice and style.

6.4 Challenges

We faced a few challenges during the course of this project. With Real time style transfer, the image transformation network needed to be trained on Microsoft's COCO dataset that has 80000 training examples and training needs to be done for each of the style images. The process took about 6 hours for each style. Since we used DSMLP GPU clusters, we faced timeout errors frequently and had to restart the training process from the saved checkpoint. To overcome this, during the later part of the project, we ran the training process as background jobs, which helped the cause to some extent.

6.5 Future Work

As discussed in the motivation section, style transfer can be extremely useful in the domains of image/video processing, and computer graphics. There is active research in the applications of image stylization methods to photorealistic style transfer, object rendering based on example styles. A prominent characterizer - Facestyle/Puppetron that uses example based synthesis of stylized facial animation for character animation was released by Adobe in 2017. Numerous new extensions to these ideas are being published regularly. The style of the image can more or less be learnt by taking into consideration a small patch of the image; Example - style characteristics such as brush strokes and painting medium (water-based, oil-based, charcoal etc.) that determine the smoothness and grain-size. For image stylization tasks such as transforming photographed images in to painted styles, this generalization of the style of a patch as the style of the entire image, greatly reduces the required computation resources and time. Also, the level of style features needed for this application are primitive and can be captured by fewer layer architecture. Future work could explore more upon application based optimizations.

7 Appendix

7.1 BitBucket Repo Link

All the files used to generate the the results described in this project are available at this bitbucket repository. https://bitbucket.org/schittam20101995/ece285$_r embrannts/src/master/$

7.2 BitBucket Repo Readme

Description: This is project B - Style Transfer developed by team RembraNNdts composed of Sahiti, Sushmitha, Richa and Mahima. Our group aims at implementing the style transfer by replicating the experiment proposed by Gatys et al. in 2015. We also implement the style transfer by using Real Time Style Transfer approach.

File organization:

- Neural Style Transfer – Folder contains code for neural style transfer proposed by Gatys et al.
  - demo.ipynb – Run a demo of the code
  - train.ipynb – Run an example of the code
  - utils.py – Module for model, methods and dataset
  - visu.py – Module for visualizing images and results
- Real-time Style Transfer – Folder contains code, resources and results for real-time style transfer
  - Code - Folder contains code for real-time style transfer
    - demo_rtst.ipynb – Run a demo of the code
    - train_rtst.py – Run the program for loading dataset, training saving a model
    - architecture.py – Module for classes
    - utils_rtst.py – Module for visualizing results
    - nntools_RTST.py – Module for implementing checking point for training
    - output_generator.ipynb – Saves content, style and transformed images
  - Models - Folder contains trained style models
    - transform_monet – Claude Monet's Oak fontainebleau style
    - transform_starry – Vincent Van Gogh's Starry night style
    - transform_irises – Vincent Van Gogh's Irises style
    - transform_cezanne – Paul Cezanne's forest style
  - Images - Folder contains style, content, transformed images (naming convention: style images have prefix 's_', content images have prefix 'c_', transformed images: style name + content image name)

References

[1] Gatys LA, Ecker AS, Bethge M. A Neural Algorithm of Artistic Style; 2015. Cite arxiv:1508.06576. Available from: `https://arxiv.org/abs/1508.06576`.

[2] Johnson J, Alahi A, Fei-Fei L. Perceptual losses for real-time style transfer and super-resolution. In: European Conference on Computer Vision; 2016. Available from: `https://arxiv.org/pdf/1603.08155.pdf`.

[3] Long J, Shelhamer E, Darrell T. Fully Convolutional Networks for Semantic Segmentation; 2015. Cite arxiv:1411.4038. Available from: `https://arxiv.org/abs/1411.4038`.