Comparison of Traditional & Modern Style Transfer Methods

Annabelle Kanchirathingal
CIVE 6358
University of Houston
ajkanchi@cougarnet.uh.edu

Abstract

Style Transfer, an interesting domain in computer vision, combines the content of one image with the style of another to create artistic outputs. Today there are many style transfer techniques, each with their own strengths & limitations. Which is why it is essential to compare methods and identify where these techniques can improve and the specific tasks that they would be useful for. This project presents a comparative analysis of traditional & modern methods of style transfer techniques. The style transfer techniques focused on in this project are Neural Style Transfer (NST) technique by Gatys et al [11] and the Adaptive Instance Normalization (AdaIN) technique by Huang et al [2]; both utilizing the VGG-19 neural network for feature extraction. Quantitative metrics such as total loss, visual quality, alongside human evaluation, highlight the strengths and limitations of each method. The findings revealed that NST excels in intricate style representation, with a low total loss; but requires significant time for iterative optimization during training. Whereas AdaIN offers greater adaptability and faster stylization; but fails at accurately translating style while maintaining content, accounting for its larger total loss values.

1. Background

1.1. Introduction

The problem of generating artistic images using computational methods has gained substantial traction, driven by advancements in algorithms that enable unique approaches to style transfer. While modern methods of style transfer continue to evolve, revisiting traditional approaches remain valuable. These earlier techniques may still deliver superior results or achieve comparable computational performance; offering insights that can inform the development of future methods.

This project presents a comparative study of two style transfer methods: the VGG-19 based neural network (NST) and the VGG-19 based Adaptive Instance Normalization (AdaIN) model.

This project was inspired by a student led project from Stanford University, which explored style transfer using CycleGAN and NST with VGG-19 [1]. Their findings depicted that the images generated by their NST model translated style better than the image generated by their CycleGAN model. I sought to further these findings, which is why I chose to compare their best model (NST with VGG-19) with a more modern technique of style transfer (AdaIN).

The results found in this project demonstrate that while AdaIN offers faster stylization, the traditional neural style transfer method still outperforms it in translating style while maintaining content structure, even though it is more computationally expensive. Figure 1 depicts an example of the images produced by either model. Table 1 depicts the average time it takes either model to stylize a single image.

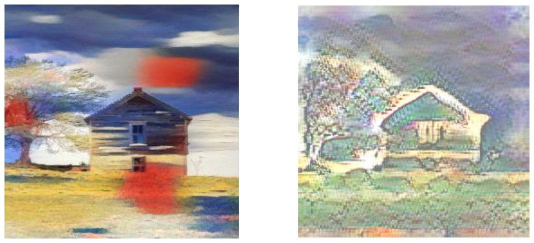

Figure 1: *The image on the left is stylized using the NST with VGG-19 model. The image on the right is stylized using AdaIN with VGG-19. The images are supposed to emulate the style found in Van Gogh's Starry Night.*

Model	**Time (seconds)**
NST with VGG-19	2569.23 seconds
AdaIN with VGG-19	0.86 seconds

Table 1: *Time taken in seconds for either model to stylize a single image*

1.2. Related Work

The Convolutional Neural Network (CNN) [11] Gatys was the first to introduce using a neural network for style transfer and was able to successfully generate an image accurately capturing style. However, this method was very slow due to its iterative optimization of the generated image. However, by using VGG-19 to extract features in the neural style transfer model, it led to a much faster stylization of images.

Instance Normalization (IN) [2] IN normalizes the mean and standard deviation across spatial dimensions for each sample. AdaIN matches the mean and standard deviation of the content feature map to that of the style feature map. This ideally allows AdaIN to precisely translate style while retaining the semantic structure of the content image.

Batch Normalization (BN) [2] BN normalizes the mean and standard deviation of a batch for each feature channel Since BN averages feature statistics across the batch, it loses instance specific information. Its' performance also suffers when the batch sizes are too small.

2. Data

All datasets for this project were sourced from Kaggle. This study utilized Kaggle's *Best Artworks of All Time* for training and testing for style [5]. For training and testing for content, Kaggle's *Monet2Photo* was used [7]. The former provides paintings by many artists contributing to various styles and the latter provides photos of natural landscapes that will be used for content. Although the style dataset provided artwork from multiple artists, many of the images were of low quality and did not depict images of landscapes. To give the AdaIN model the best chance at transferring style accurately, since it has the potential of performing poorly if the content and style image are too different in terms of structure; I decided to manually select 500 of the best quality artworks that depict landscapes from this dataset. The content dataset had a similar problem, along with the fact some of the images were not actually of landscapes but of people, so I once again manually selected 500 images from the content dataset. Figure 1 shows an example of images sourced from the content and style datasets. For preprocessing I resized the images to 512x512 pixels; and normalized the images to have a mean of [0.485, 0.456, 0.406] and a standard deviation of [0.229, 0.224, 0.225]. Preprocessing the images in this manner allows for consistent feature extraction using the VGG-19 neural network.

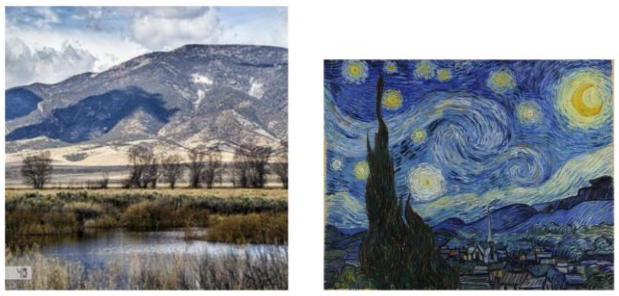

Figure 2: *The image on the left was sourced from the content dataset. The image on the right was sourced from the style dataset.*

3. Methods

3.1. VGG-19 Neural Network

VGG-19 is a deep convolutional neural network that was developed by the University of Oxford; it has 19 layers and was trained on 1.2 million images from the ImageNet database [6]. It utilizes 3 x 3 convolutional filters stacked with increasing depth, along with five max-pooling layers to downsample spatial dimensions, and three fully connected layers (fully connected layers are not utilized in either model as it is not required for feature extraction) [9]. The deeper layers of the network can capture semantic information, such as object shapes. The lower layers of the network can capture low-level features like edges, texture, patterns, and colors.

3.2. Neural Style Transfer with VGG-19

In NST, VGG-19 works as a feature extractor, and extracts the feature maps for the content, style, and generated image. Deeper layers of the VGG-19 network are assigned to the content layer, while lower layers of the network are assigned to the style layer. These feature maps are then used to calculate content loss, style loss, total variation loss, and total loss (Loss function formulas are detailed in 3.2.2.).

For the development of my NST model I relied on TensorFlow's tutorial on neural style transfer [12]. I used their implementation as more of a guide rather than copying the code they provided directly. I followed Gatys et al's foundational paper on NST to implement loss computation classes and select specific layers of the VGG-19 network as content and style layers [11].

3.2.1. Architecture

The architecture of my NST model is centered around the VGG-19 network. The content layer is conv4_2, and the style layers include conv1_1, conv2_1, conv3_1, conv4_1, conv5_1. The style layers are the same ones utilized in Gatys et al's implementation of NST. Only the convolutional and max-pooling layers of the VGG-19 network are used, and all parameters of the network are frozen to ensure the network works exclusively for feature extraction.

The process begins by initializing a generated image with random noise. During each epoch, the VGG-19 network will extract feature maps for the content, style, and generated image. These feature maps will then be used to compute the following loss functions: content loss, style loss, total variation loss, and total loss.

Key hyperparameters of the model include content weight, style weight, total variation weight, and learning rate. These first three hyperparameters mentioned control the contribution of each loss term to the total loss. The LBFGS optimizer is then employed to minimize the total loss, refining the generated image with each epoch.

3.2.2. Loss Functions

The loss functions for content loss (1), style loss (2), and total variation loss (3), and total loss (4) are provided below.

$$L_{content} = \frac{1}{2}\sum_{i,j}(F_{ij}^{generated} - F_{ij}^{content})^2 \qquad (1)$$

Content loss measures the MSE between the generated image feature map and content image feature map.

$$L_{style} = \sum_{l=1}^{L} w_l \times \frac{1}{4N_l^2 M_2^l} \sum_{i,j}\left(G_{ij}^{gen,l} - G_{ij}^{style,l}\right)^2 \qquad (2)$$

Style loss measures the gram matrix difference between the generated image and the generated image.

$$L_{TV} = \sum_{i,j}\left(\left(I_{i+1,j} - I_{i,j}\right)^2 + (I_{i,j+1} - I_{i,j})^2\right) \qquad (3)$$

Total Variation Loss computes the sum of squared differences between adjacent pixels, both horizontally and vertically.

$$L_{total} = \alpha \cdot L_{content} + \beta \cdot L_{style} + \gamma \cdot L_{TV} \qquad (4)$$

Total Loss computes the weight of each hyperparameter multiplied by the loss for the respective hyperparameter.

3.3. AdaIN with VGG-19

In AdaIN, the VGG-19 network works similarly to NST, where it is used for extracting features. However, instead of assigning specific layers to content and style, its layers are used to form the encoder. The extracted feature maps from the encoder are then transformed using the AdaIN class (1). Then the decoder (an inverted version of the VGG-19 based encoder) will decode the stylized image and then passed through the encoder once more to calculate content loss (2), style loss (3), and total loss (4).

For the implementation of the Adain model, I utilized an existing GitHub implementation of Huang et al's Adain model [13] (All credit for this model and its implementation go its authors: Anthony Ke, Hersh Vakharia, Issac Fung, Ali Baker). To stray away from copying the code directly, I made modifications to the AdaIN class following Huang et al's foundational paper and restructured the layers for the encoder and decoder and adjusted the hyperparameters as needed. My initial implementation of the AdaIN model proved to be unfruitful, which is why I decided to use an existing repository with some modification.

3.3.1 Architecture

Like my NST model, the architecture for the AdaIN model is also centered around the VGG-19 network. Only the convolutional and max-pooling layers are used from the VGG-19 network, and its parameters are frozen so that only the decoder is optimized. In the VGG-19 encoder, the first 21 layers of the network are used. The content layer includes relu4_2 and the style layers include relu1_1, relu2_1, relu3_1, and relu4_1, this differs from the style layers used in GitHub repository. Forward hooks are utilized to capture the feature maps at the layers mentioned above. Once the feature maps are extracted, they are passed through the AdaIN class (1) which will then normalize the content feature maps mean and standard deviation to match that of the style feature map.

$$AdaIN(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y) \qquad (1)$$

Once the image is transformed using the AdaIN class, it is then passed to the decoder, which is the inverse of the VGG-19 encoder. The convolutional layers of the decoder use a padding of 1 and reflection padding which helps preserve edges. The ReLU layer introduces non-linearity into the model. The Upsampling layer increased the spatial dimensions; here I decided to change mode from nearest to bilinear, because having the mode as nearest resulted in blocky generated image. After the decoder decodes the stylized image, it is returned and then passed back to the encoder to compute content loss (2), style loss (3), and total loss (4) (The GitHub repository had used average loss to evaluate the model, but for accurate comparison between both models I used total loss).

Key hyperparameters of the model include content weight, style weight, and learning rate. These hyperparameters control the contribution of each loss term to the total loss. The Adam optimizer then minimizes the calculated total loss and optimizes the parameters of only the decoder.

3.3.2. Loss Functions

$$L_{content} = \frac{1}{2}\sum_{i,j}(F_{ij}^{generated} - F_{ij}^{content})^2 \qquad (2)$$

Content loss measures the MSE between the generated image feature map and content image feature map.

$$L_{style} = \sum_{l=1}^{L} w_l \times \frac{1}{4N_l^2 M_2^l} \sum_{i,j}\left(G_{ij}^{gen,l} - G_{ij}^{style,l}\right)^2 \qquad (3)$$

Style loss measures the gram matrix difference between the generated image and the generated image.

$$L_{total} = \alpha \cdot L_{content} + \beta \cdot L_{style} \qquad (4)$$

Total Loss computes the weight of each hyperparameter multiplied by the loss for the respective hyperparameter.

4. Experiments

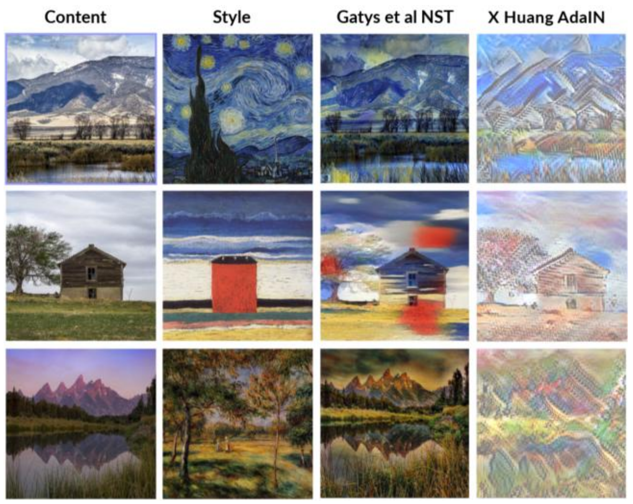

Figure 3: *Photo grid of NST & AdaIN generated images compared to their respective content and style images*

4.1 NST with VGG-19 Training

Initially when I first began implementing and training the model I only used content weight, style weight, and learning rate as hyperparameters and utilized the Adam optimizer to minimize total loss. However, this resulted in the image generated becoming overly saturated, figure 4 depicts an example of a generated image using the Adam optimizer. This issue is likely due to the adaptive learning rate mechanism of the optimizer, which can amplify certain gradients disproportionately. During this time, training a single content image and style image took around 4 hours. After further research on optimizers, I decided to use the LBFGS optimizer. When I switched to the LBFGS optimizer it resolved both problems of overly saturated images and long training time. Using the LBFGS optimizer resulted in a stable convergence, which produced more visually balanced images and a shorter training time for my model.

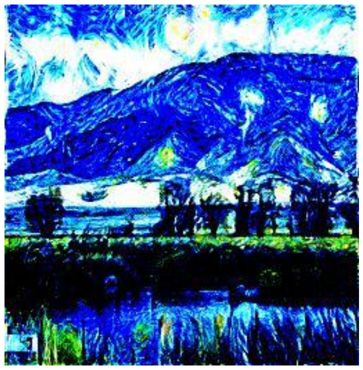

Figure 4: *Saturated generated image using Adam Optimizer*

After switching the optimizer, I recognized another problem with the generated images. The generated images seemed to be transferring style to each pixel, resulting in the generated image looking more pixelated. After further research, I decided to implement total variation loss, which penalizes adjacent pixels, both horizontally and vertically. The implementation of total variation weight and total variation loss generated a more cohesive image and lowered content loss.

For experimentation, I manually tuned each hyperparameter. The first experiment relied on using a lower style weight (style weight set to 1) compared to content weight (content weight set to 30). This resulted in the generated image capturing all the content structure but transferring none of the style. For experiment 2, I set style weight to 1000, content weight to 10, and total variation weight to 0.0001. The image generated from this hyperparameter configuration captured the style image better but was still highlighting content structure over style transfer, as my style loss was very high. For my third experiment, I set style weigh to 100000, content weight to 1, and total variation weight to 0.000001. This configuration of hyperparameters was like the hyperparameters used in Gatys et al's paper which resulted in precise style transfer while maintaining content structure. It resulted in a generated image that was smooth, and accurately transferred style. The total loss, content loss, and total variation loss for images trained with these hyperparameters were slowly converging to 0. Figure 3 depicts generated images using experiment 3's configuration of hyperparameters. Figure 5 depicts the total loss, content loss, and total variation loss for a single content and style image (specifically the image trained to transfer style from Van Gogh's Starry Night seen in figure 3).

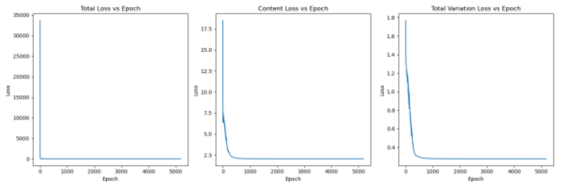

Figure 5: *Total Loss, Content Loss, and Total Variation Loss by epoch for NST*

The NST model with the optimal hyperparameter configuration trains with a single content and style image and takes an average of 2569.23 seconds (46 minutes) to return a stylized image, about 8.1421 seconds per epoch. Although this is an improvement from the 4 hours of training time seen before, it is still more computationally expensive than the AdaIN model. However, even though it does require longer training time, it provides images that can capture artistic style like color, texture, brushstrokes all while maintaining content structure.

4.2 AdaIN with VGG-19 Training

To train the AdaIN model I manually tuned the following hyperparameters: content weight, style weight, learning rate, and batch size. Since training this model was computationally expensive, I decided to first try tuning my hyperparameters and train the model on a few epochs to see how the loss values are converging.

For experiment 1, I set the content weight to 20, style weight to 1, batch size to 8, and learning rate to 0.001. With this configuration I saw a high style loss, which did not seem to decrease, as the weight I set for style was too low. For experiment 2, I set the content weight to 15, style weigh to 20, batch size to 8, and learning rate to 0.0001. This resulted in a high style loss and a low content loss. To see if this configuration of hyperparameters generated good images, I trained the model on a 100 epochs. The training time for this experiment took 25 hours because the model was training on 500 images for both content and style. I then saved the best model (model which resulted in the lowest total loss) and used it to test on a single image for content and style. The generated image from this configuration was not able to capture style well, but did maintain content structure and accurately capture the colors found in the style image. Also, the time required to test the model was drastically lower than the time it took to train, around 0.67 seconds. The style loss for this experiment was also quite high and did not seem to converge. For experiment 3, I set content weight to 1, style weight to 30, batch size to 8, and learning rate 0.000001. This configuration of hyperparameters was also trained on 100 epochs, and it took around 25 hours to train as well. During this experiment I saw content loss and style loss both decrease with each epoch. Again, I saved the best model and tested the model on a single content and style image. The resulting image created from this configuration-maintained content structure and showed better signs of transferring style. Figure 3 shows the generated images for the AdaIN model using this configuration of hyperparameters. Figure 6 shows the total loss over epochs for this configuration of hyperparameters. As you can see the loss, while it is converging smoothly, it is still relatively high at 100 epochs.

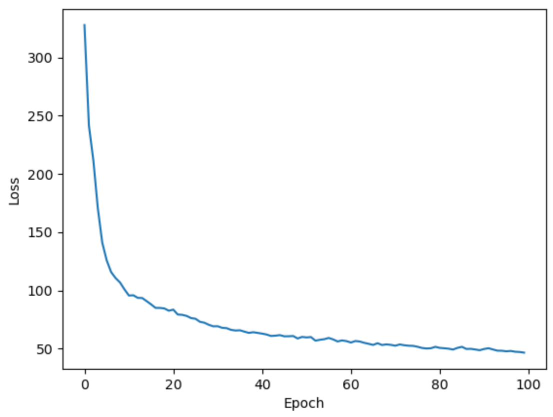

Figure 6: *Loss plot for AdaIN with hyperparameters: content weight = 1, style weight = 30, batch size = 8, learning rate = 0.0000001, epochs = 100*

While training the AdaIN model took longer, testing the best model on a sample of content and style images took significantly less time than is required to train the NST model. Additionally, the AdaIN model proved it is flexible enough to be trained and tested on a variety of styles; the same cannot apply to the NST model as it can only stylize a single content and style image at one time. This could have also been its downfall, as the reason why the generated images fail to transfer style properly could be since some style images dominate over others, leading the model to have bias towards some styles. While my comparison is not enough, I decided to also use human evaluation to compare the generated images of the two models (detailed in section 4.3)

4.3 Human Evaluation

For human evaluation, the data was collected from 30 University of Houston students, from varying backgrounds. I created a google form which prompted the user with six questions. Every two questions were prefaced by three images shown to the user; the first image was the artwork (labeled with the artists name & title of their artwork), the second image (labeled as "*Image A*") was the generated image by the NST model, and the third image (labeled as "*Image B*") was the generated image by the AdaIN model. The questions asked after the images were presented were 1: "*How well does Image A emulate* [Artist Name's] *art style*" and 2: "*How well does Image B emulate* [Artist Name's] *art style*". For the answers, the user could pick a number between 1 and 5, 1 being the least like the artist's style and 5 being the most like the artist's style.

The images selected for each question are the same as the images found in figure 3. Every image generated by the NST model was scored higher by users compared to the images generated by the AdaIN model. The full results of the human evaluation are shown in table 2.

Style Images	NST score	AdaIN Score
Vincent Van Gogh, *Starry Night*	Highest Rating: 39% voted 5 (most like style image) Lowest Rating: 21.7% voted 2	Highest Rating: 47.8% voted 1 (least like style image) Lowest Rating: 13% voted 4
Kazimir Malevich, *Red House*	Highest Rating: 26.1% voted 3 Lowest Rating: 4.3% voted 1	Highest Rating: 56.5% voted 1 Lowest Rating: 8.7% voted 2
Pierre Auguste Renoir, *The Glade*	Highest Rating: 47.8% voted 5 Lowest Rating: 13% voted 3	Highest Rating: 52.2% voted 1 Lowest Rating: 4.3% voted 5

Table 2: *Results of human evaluation*

5. Novel Insight

Surprisingly the traditional NST model with VGG-19 outperformed AdaIN with VGG-19 in terms of translating style, color, texture, and patterns while maintaining the structure of the content image. The AdaIN model struggles to accurately transfer style, and only managed to transfer color and minimal texture while maintaining the structure of the content image. The NST model also showed lower loss values for content loss and style loss compared to the loss values produced by the AdaIn model.

Despite its innovative approach to style transfer, normalizing style statistics to adaptively adjust the style transfer; the NST model showed a more accurate reproduction of style while preserving content structure.

This particular finding is novel, because it suggests that, while AdaIN maybe seen as the more modern approach to style transfer; the traditional NST still triumphs when it come to capturing detail and preserving content. This result challenges assumptions about the supremacy of modern methods of style transfer and provides valuable insight into the trade-offs between style adaptation and computational complexity.

6. Conclusion

In conclusion my experiments comparing NST with VGG-19 and AdaIN with VGG-19 have provided valuable insights into the strengths and weaknesses of each approach. I learned that while NST is only able to translate a single style in each training process and is more computationally expensive, it produces images that can capture style well while maintaining content structure. NST with VGG-19 is suitable for tasks which require a detailed generated image that can capture style well, where there are computational resources available to handle the processing power it requires. Furthermore, I learned that AdaIN with VGG-19, while it does not produce the most aesthetically pleasing images, its flexibility and ability to stylize multiple images in a single pass allow it to be suitable for tasks in which multiple styles need to be transferred without attention to the detail of the generated image.

For future work, I would like to explore the creation of a hybrid model of the two. I believe a model such as this would combine the strengths of both approaches and result in stylized images that match generated images produced by NST, while stylizing them in a single pass like AdaIN. Additionally, I would like to explore other pretrained networks along with VGG-19 on both models. I believe using something like ResNet with VGG-19, would allow for the extraction of more intricate features that could result in a better generated image.

References

[1] Pan, J., Kao, C., & Yuan,. X. (2020). "Paint like Vincent Van Gogh: Artistic Style Generator Using CycleGAN and VGG19. http://cs230.stanford.edu/projects_fall_2020/reports/55792990.pdf

[2] Huang, X., Belongie, S., & Karras, T. (2017). Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. arXiv. https://arxiv.org/pdf/1703.06868

[3] Y. Tao, "Image Style Transfer Based on VGG Neural Network Model," 2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), Dalian, China, 2022, pp. 1475-1482, doi: 10.1109/AEECA55500.2022.9918891.

[4] Len Du; Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2020, pp. 3150-3159

[5] Best Artworks of All Time. https://www.kaggle.com/ikarus777/best-artworks-of-all-time, Accessed: 2024-10-16

[6] K. Simonyan, A. Zisserman Very Deep Convolutional Networks for Large-Scale Image Recognition International Conference on Learning Representations, 2015

[7] Monet2Photo. https://www.kaggle.com/balraj98/monet2photo Accessed: 2024-10-16

[8] AdaIN model. https://github.com/xunhuang1995/AdaIN-style

[9] "Vgg19 — Torchvision Main Documentation." Pytorch.org, 2023, pytorch.org/vision/main/models/generated/torchvision.models.vgg19.html.

[10] Mirzazada, Elvin. "Neural Style Transfer with Deep VGG Model - Elvin Mirzazada - Medium." Medium, 13 May 2020, medium.com/@mirzezadeh.elvin/neural-style-transfer-with-deep-vgg-model 26b11ea06b7e.

[11] Gatys, Leon A., et al. “A Neural Algorithm of Artistic Style.” *arXiv.Org*, 2 Sept. 2015, arxiv.org/abs/1508.06576.

[12] “Neural Style Transfer : Tensorflow Core.” *TensorFlow*, www.tensorflow.org/tutorials/generative/style_transfer. Accessed 9 Dec. 2024.

[13] Hvak. “HVAK/Adain-Style-Transfer: A Pytorch Implementation of Adaptive Instance Normalization Arbitrary Style Transfer.” *GitHub*, github.com/hvak/adaIN-style-transfer/tree/main. Accessed 9 Dec. 2024.