CSC 871 - Deep Learning

Team Project Final Report - Style Transfer

Mark Kim, Satvik Verma, Fabian Weiland

May 19, 2024

Instructor:
Prof. Robert Mateescu

Contents

1 Abstract

Advancements in machine learning have brought generative AI into the mainstream. Unfortunately, trying to reproduce these kinds of advanced models with limited compute resources is infeasible. Nevertheless, there are ways to utilize models that may not match the state-of-the-art and still produce amazing results while learning about the foundations upon which they are built upon. We not only implement style transfer, but also mixed style transfer [1] and explore a variety of loss functions including: Gram matrix [2], Sliced Wasserstein [3], and Wasserstein-Gaussian Loss [4].

2 Introduction

Mixed style transfer is a method of using a content image and a multitude of style images to produce a final combined image. The goal of this is to maintain the essence of the content image while applying artistic styles to it. Gatys, et al. pioneered style transfer by using features extracted from pretrained Convolutional Neural Networks such as VGG-19 [2]. They found that they could separate the content and style representations from each other and produce an image that maintains the content while applying a textural style to it that matches the style image. Instead of optimizing the neural network to input images, they instead optimized the image to match the feature resprepresentations provided by the layers of the CNN. For the content loss, this was simply the mean squared error (MSE) between the feature representations of the content image and target image at a particular layer in the CNN and is noted in Gatys, et al. as:

$$\mathcal{L}_{\text{content}}\left(\vec{p},\vec{x},l\right)=\frac{1}{2}\sum_{i,j}\left(F_{ij}^{l}-P_{ij}^{l}\right)^{2}. \tag{1}$$

with $\vec{p}$ and $\vec{x}$ denoting the original and target images, with P^l and F^l being the respective feature representations of the images at layer l [2]. For the style loss, they employed gram matrices which captures the correlations of feature responses at each layer specified, which is given by

$$\mathcal{G}^{l}=\frac{1}{N^{l}}F^{l}\left(F^{l}\right)^{T} \tag{2}$$

where N^l is the number of features at layer l.

Beyond the methods laid out in Gatys, et al., we attempt to implement two more loss functions: Sliced Wasserstein and Wasserstein-Gaussian Loss. Both of these loss functions are rooted in Optimal Transport theory. Wasserstein distance, also known as "Earth-Mover's Distance", quantifies the notion of the distance between two distributions [5]. To expand upon this, we can use the anal-

ogy of having piles of dirt and holes which need to be filled from these piles. Wasserstein distance is a method of calculating the cost of moving these dirt piles into the holes as illustrated in Figure 1.

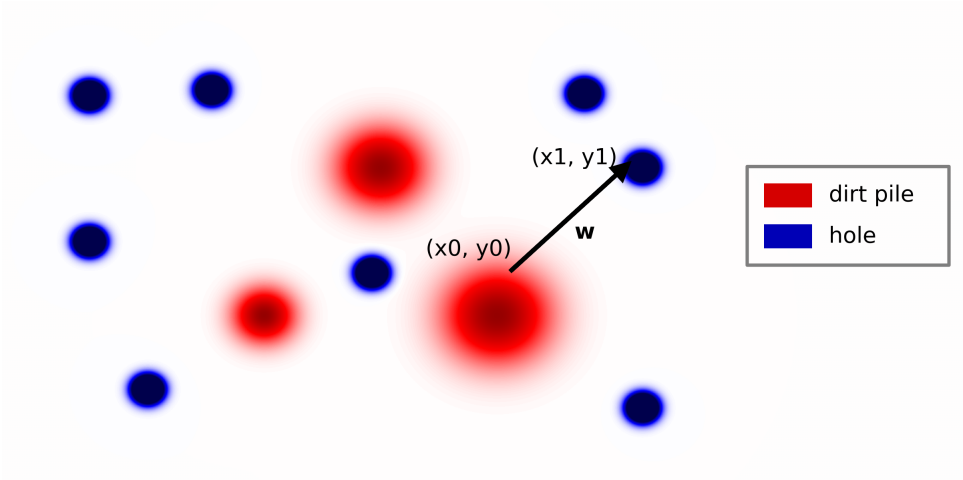

Figure 1: Wasserstein Distance

The Sliced Wasserstein method projects the features into one dimensional space (Figure 2), and the Wasserstein-Gaussian method assumes a normal distribution for the feature space. Unfortunately, the math underpinning these methods are still a bit outside our complete understanding, but the intuition applies.

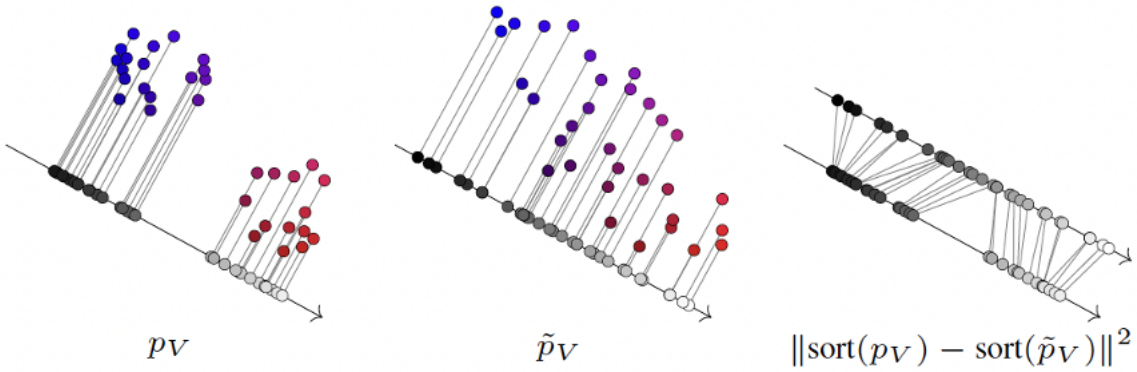

Figure 2: Sliced Wasserstein Distance

The final implementation that we wanted to investigate was mixed style transfer. We initially wanted to examine multiple style transfer, but due to time limitations, we had to abandon that in favor of two style transfer. This step required us to introduce another hyperparameter that would act as a "cross-fader" between two different style images. This style cross-fader provides our

model a way to determine the balance between what proportion of the different style images to use when calculating the total loss.

3 Methods

Our model was built from the Visual Geometry Group 19 Layer Convolutional Neural Network (VGG-19). This model has 16 CNN layers, 2 fully connected layers, and a final Softmax layer. The filters used in the CNN layers, had a kernel size of 3 x 3, stride of 1, and padding of 1. Since we were only concerned with the learned feature representations, we discarded the fully connected and Softmax layer. We then used the set of default layer weights as provided in the Torchvision models package.

3.1 Prototype Implementation

We began with implementing a minimally working prototype following the structure and steps below.

1. Image Loading: The `image_loader` function resizes images to a specified size and converts them into PyTorch tensors

2. Model Loading: It loads the VGG19 model pre-trained on the ImageNet dataset. The VGG19 model is used for feature extraction.

3. Custom Losses: Defines custom loss functions. Content loss measures the difference between the feature maps of the content image and the generated image. Style loss measures the difference in Gram matrices of the style image and the generated image.

4. Normalization: Defines a normalization module to normalize input images. This is required because the VGG19 model expects input images to be normalized with specific mean and standard deviation values.

5. Model Modification: Modifies the VGG19 model by removing its fully connected layers and adding content and style loss layers. This modified model is used for feature extraction during style transfer.

6. Optimization Setup: Sets up the optimization process. LBFGS optimizer is used for optimization.

7. Style Transfer Process: The `run_style_transfer` function combines content and style images to produce an output image that combines the content of one image with the style of another. It optimizes the input image to minimize the content and style losses.

8. Optimization Loop: The optimization loop iteratively updates the input image to minimize the combined content and style losses. It utilizes the LBFGS optimizer to update the input image.

9. Output Visualization: Finally, it visualizes the output image using Matplotlib. The generated image is displayed after the style transfer process is completed.

3.2 Mixed-Style Transfer Implementation

Once we completed the implementation of our prototype, we moved ahead with expanding and generalizing our model and code. Here, we enumerate the changes we made to our prototype to improve extensibility and allow us to insert different loss functions into our model.

1. Image Handling and Setup:

   (a) Image Size Selection: Determines the size of the images for processing, based on the available computational resources. Larger sizes provide higher resolution but require more memory and computation time.

   (b) Directory Setup: Defines paths to directories containing input content and style images, as well as the directory for saving the output images.

2. Model Loading and Layer Retrieval:

   (a) Pre-trained VGG19 Model: Loads the VGG19 model pre-trained on the ImageNet dataset. VGG19 is commonly used in style transfer tasks due to its effectiveness in capturing image features.

   (b) Layer Retrieval Function: Defines a function to extract the names of convolutional layers from the VGG19 model. These layer names are used to specify content and style layers for feature extraction during style transfer.

3. Model Building and Loss Functions:

   (a) Model Construction: Defines a function to build a mixed style transfer model. This function modifies the VGG19 model by inserting content and style loss layers at specified layers.

(b) Loss Functions: Specifies loss functions for content and style. Content loss measures the difference in content between the content image and the generated image. Style loss measures the difference in style between the style image and the generated image. Different loss functions (e.g., `GramMatrixLoss`, `SlicedWassersteinLoss`) are provided for style loss computation, allowing flexibility in style transfer.

4. Image Generation:

   (a) Image Optimization: Defines a function to generate the mixed style image. It sets up an optimization process to iteratively update the output image to minimize the combined content and style losses.

   (b) Closure Function: Defines a closure function for optimization. This function computes the total loss, which is a combination of content and style losses, and performs backpropagation to update the output image.

5. Main Function:

   (a) Input Image Loading: Loads content, style, and second style images from specified directories.

   (b) Model Setup: Sets up model parameters including content and style layers, loss functions, and optimization parameters.

   (c) Style Transfer Execution: Executes style transfer using different loss functions. It generates mixed style images by blending styles from two different images onto the content image.

   (d) Output Saving: Saves the generated images and associated metadata for analysis and further processing.

6. Execution:

   (a) Function Invocation: Calls the main function to initiate the style transfer process.

   (b) Image Visualization and Saving: Visualizes the input images, generated mixed style images, and saves the results to specified output directories.

This completes the implementation and the working code of our Neural Style Transfer model.

4 Results

In this chapter, we will present the results of applying our deep learning neural style transfer to various images. By blending the content of one image with the style of another image, our model

produces visually unique outputs. We will evaluate the performance and quality of these images. Additionally, we will examine a mixed style transfer example, which involves utilizing one content image and two style images to generate an output image that combines the content and style of both style images. At the end we will provide an example demonstrating the use of three different loss functions on the same content and style images.

Figure 3: Noise input, content, style, and output images from left to right

Figure 4: content input, content, style, and output images from left to right

We will discuss two methods for presenting and producing the output image from our neural single style transfer model. In the first method we start with a noise image (with randomized pixel values) and apply content loss and style loss to iteratively update the input pixel values, aiming to match the content of the content image and the style of the style image (Figure 3). In the second method we use the content image as the input image and update its pixel values until the desired style is blended in (Figure 4).

Both methods used the same hyperparameters to produce the output image, with the only difference being the content loss weight. In method 1, we used a content loss weight of 1000, while in method 2, we used a content loss weight of 10. The reason for this is straightforward: since the input image in method 2 already contains the content, less content induction is needed compared to method 1, which starts with a noise image as the input. It is clear that the output image from method 2 is superior to that from method 1. Additionally, method 2 requires less computational power, delivering a superior output image more quickly.

With an increased number of epochs from 50 to 2000, method 2 produces the result shown in Figure 5.

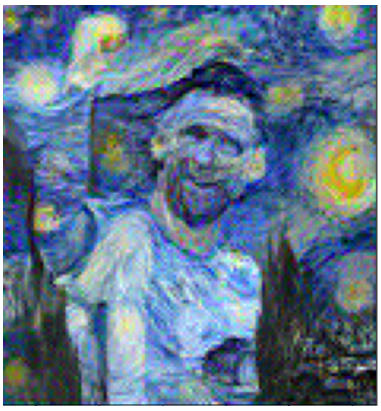

Figure 5: 2000 epochs

Figure 6: content, style 1, and style 2 images from left to right

Next we will look at a mixed style transfer of the images in Figure 6.

The images in Figure 7 display the resulting images ranging from the ”cross-fader” values, λ, from 0 to 1 in 0.1 value increments. This style cross-fader provides our model a way to determine the balance between what proportion of the different style images to use when calculating the total loss.

Finally, we present the output images generated using the three different loss functions applied to the same content and style images in Figure 8.

Each loss function produces distinct output images. For the hyperparameters used in this example, the Gram matrix and Sliced Wasserstein loss functions yield better results than the L2 Wasserstein Gaussian. However, none of the three loss functions produce perfect outputs. Due to limited time and computing power, we were unable to optimize the hyperparameters for all the loss functions so that we get perfect results. Nevertheless, this still demonstrates the differences in outputs for the three loss functions, resulting in decent style transfer images.

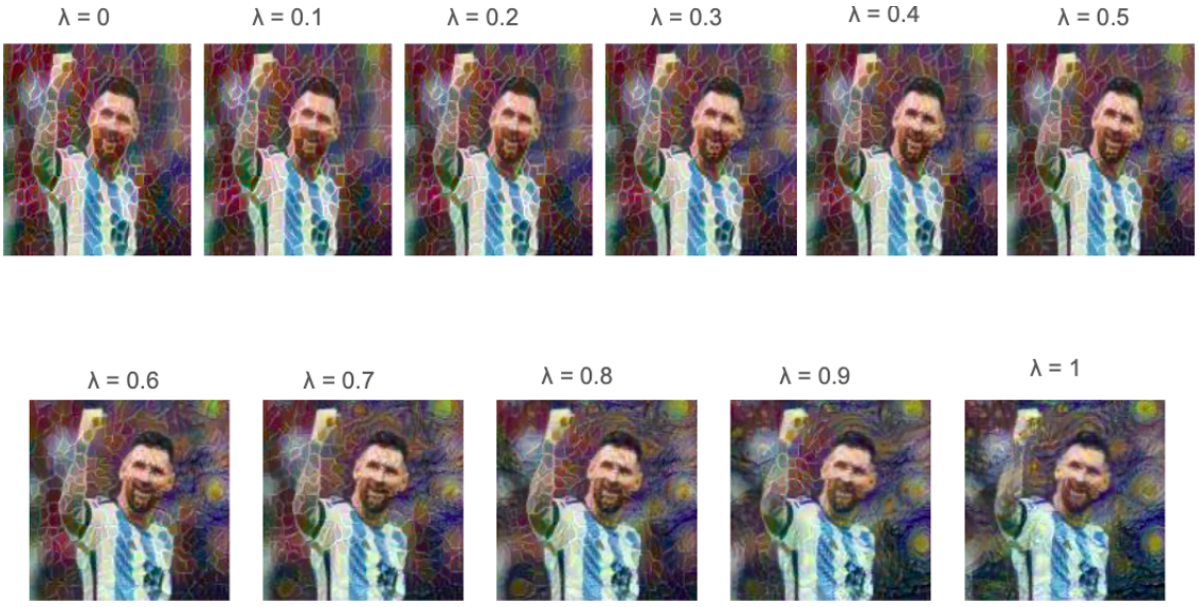

Figure 7: Mixed style transfer with λ values ranging from 0 to 1

5 Discussion

5.1 What We Learned

Through this project, we gained a deep understanding of neural style transfer models and its various implementations. Since first we tried building and training our own CNN model, we realized how compute intensive this task can be and later we learned how to leverage the pre-trained VGG-19 model network to extract and manipulate feature representation of images. The process of optimizing the input image to match both content and style representations illuminated the intricacies of balancing multiple loss functions. Implementing the Gram matrix-based style loss provided us with a solid foundation in texture synthesis, while exploring Sliced Wasserstein and Wasserstein-Gaussian Loss deepened our knowledge of optimal transport theory and its applications in neural networks.

Working on mixed style transfer, we realized the importance of hyperparameter tuning, especially in controlling the cross-fading between multiple styles. This exercise highlighted how different artistic styles could be blended effectively, resulting in unique and visually appealing outputs. Additionally, we observed that starting with a noise image versus the content image significantly influenced the convergence behavior and quality of the final output.

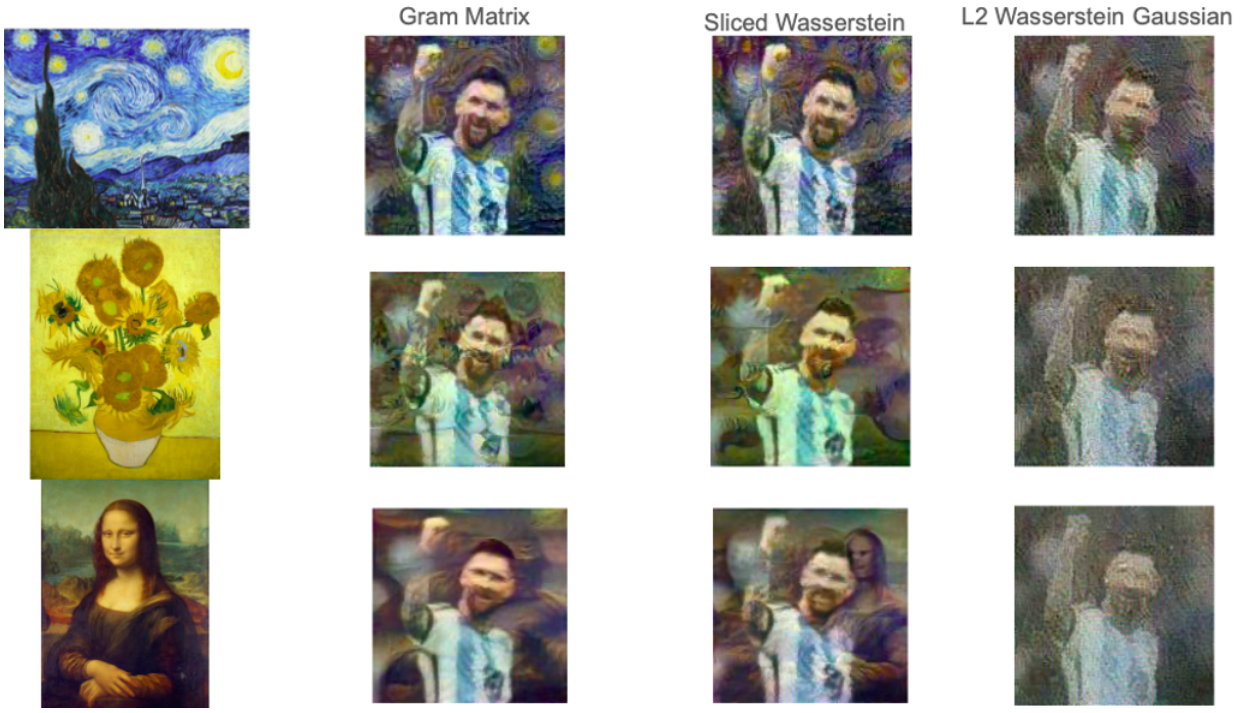

Figure 8: Results from different loss functions

5.2 Areas For Improvement

While our implementation achieved notable results, several areas could be improved. Firstly, the computational efficiency of our model could be enhanced. The iterative optimization process, particularly when starting with a noise image, was time-consuming. Implementing more efficient optimization algorithms or leveraging hardware accelerations, such as GPUs or TPUs, could mitigate this issue.

Another area for improvement is our understanding and application of the Sliced Wasserstein and Wasserstein-Gaussian Loss functions. Although we grasped the intuition behind these methods, a more profound mathematical understanding would enable us to fine-tune these losses better and achieve more consistent results.

The model's ability to handle more than two style images for mixed style transfer was limited by time constraints. Future work could explore extending the cross-fading mechanism to accommodate multiple style images, providing greater flexibility and creative control.

5.3 Future Work

Future work can focus in several promising directions such as:

1. Deepening Mathematical Understanding -
   A more thorough study of optimal transport theory and its variants can help refine the im-

plementation of Sliced Wasserstein and Wasserstein-Gaussian Loss, leading to better results. Likewise, it could provide us other avenues of exploration for other loss functions that are based upon optimal transport theory.

2. Multiple Style Images -
   Extending the mixed style transfer to handle more than two style images will offer greater creative possibilities. This could involve developing new algorithms for seamlessly blending multiple styles.

3. Real-Time Style Transfer -
   Exploring techniques for real-time style transfer, such as using feed-forward networks instead of iterative optimization, can make the technology more accessible for various applications, including video and interactive media.

4. Enhanced Optimization Techniques -
   Implementing advanced optimization techniques or leveraging more powerful hardware can significantly speed up the style transfer process, making it more practical for real-time applications.

By addressing these areas and building upon our current work, we can further advance the field of neural style transfer, making it more efficient, versatile, accessible and hopefully try out even more complex custom loss functions.

References

[1] Paul Froehling. *Mixed neural style transfer with two style images*. Dec. 2021. URL: https://towardsdatascience.com/mixed-neural-style-transfer-with-two-style-images-9469b2681b54 (visited on 05/19/2024).

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style." In: *CoRR* abs/1508.06576 (2015). arXiv: 1508.06576. URL: http://arxiv.org/abs/1508.06576.

[3] Eric Heitz et al. "Pitfalls of the Gram Loss for Neural Texture Synthesis in Light of Deep Feature Histograms." In: *CoRR* abs/2006.07229 (2020). arXiv: 2006.07229. URL: https://arxiv.org/abs/2006.07229.

[4] Vincent Marron. URL: https://github.com/VinceMarron/style_transfer/blob/master/style-transfer-theory.pdf (visited on 05/19/2024).

[5] Alex Williams. *A Short Introduction to Optimal Transport and Wasserstein Distance*. 2020. URL: https://alexhwilliams.info/itsneuronalblog/2020/10/09/optimal-transport/ (visited on 05/19/2024).