

Lab 2: Neural Style Transfer with AdaIN

ELEC 475

Prof. Michael Greenspan

Monday, September 25th, 2023

Mile Stosic (20233349)

Kieran Cosgrove (20226841)

Table of Contents

1. Introduction.....	1
2. Hardware Description	1
3. Training.....	2
3.1 Hyper-parameters.....	2
3.1.1 Gamma.....	3
3.2 AdaIN Implementation.....	3
3.3 Execution Time	4
4. Results.....	4
4.1 Loss Curve Comparisons	4
4.2 Test Comparisons.....	5
5. References.....	10

Table of Figures

Figure 1: 1k Loss Curve.....	5
Figure 2: 10k Loss Curve.....	5

Table of Tables

Table 1: Original Images used in Experiment.....	6
Table 2: 81 x Andy_Warhol_97	7
Table 3: 81 x Brushstrokes.....	7
Table 4: 81 x Chagall_Marc.....	8
Table 5: Lena x Andy_Warhol	8
Table 6: Lena x Brushstrokes.....	8
Table 7: Lena x Chagall_Marc.....	9

1. Introduction

Style transfer has garnered significant attention in the machine learning community, thanks to its ability to create visually striking and artistically enriched images. One of the pivotal techniques in this realm is AdaIN, which aligns the mean and variance of the content image with that of the style image, thus transferring the stylistic features effectively.

The primary aim of this lab report is to dissect the workings of an AdaIN style transfer network, providing insights into its components, functionalities, and performance metrics. We utilize a pre-defined architecture comprising of `AdaIN_net.py`, `custom_dataset.py`, and `test.py` modules, along with a pre-trained encoder (`encoder.pth`) to facilitate this investigation.

Our experimental journey using the COCO datasets as content and Wikiart as Style. We began with debugging on smaller “1k” datasets, ensuring that our code is robust and efficient. We then escalated our experiments to the “10k” datasets, unleashing the full potential of our AdaIN network on a desktop workstation equipped with a GEFORCE GTX 1650 GPU. The transition from a smaller to a larger dataset allows us to meticulously observe and compare the network's performance, understand its scalability, and gauge its runtime efficiency.

To provide a holistic view of our network's performance, we meticulously plot the loss, encapsulating the content loss, style loss, and the total loss for both “1K” and “10K” training processes. This graphical representation serves as a critical tool for assessing the network's learning trajectory and optimizing its parameters. Furthermore, we demonstrate the network's prowess by generating output images for a selection of content and style image combinations, illustrating the network's capacity to seamlessly blend various artistic styles with diverse content.

This report is structured to guide the reader through the various facets of the AdaIN style transfer network, beginning with this introduction and followed by detailed sections on the network architecture, dataset, training process, loss computation, results, and conclusion. Each section is crafted to provide clarity, insights, and a comprehensive understanding of the AdaIN style transfer mechanism.

2. Hardware Description

For the purpose of training our AdaIN style transfer model, we utilized a local laptop computer equipped with the NVIDIA GeForce GTX 1650 GPU. This graphics processing unit (GPU) belongs to NVIDIA's 16 series, based on the Turing architecture, and is well-regarded for its balance between performance and power efficiency, making it a suitable choice for demanding computational tasks such as deep learning and style transfer.

Choosing a local GPU for our experiments, as opposed to cloud-based platforms like Google Colab, was influenced by several considerations. To begin, utilizing a local GPU ensures uninterrupted access to the hardware resources, without being subject to time limitations or potential disconnections that can occur with cloud-based platforms. This is particularly crucial for extensive training sessions that span several hours, as was the case with our “10k” dataset which took approximately 10 hours to complete. Moreover, local processing negates the dependency on a stable and fast internet connection, which is required when using cloud-based solutions. This ensures that our experiments can run smoothly without being hampered by potential network issues. Running the model locally also provides a more hands-on learning experience, offering the opportunity to delve deep into the debugging and optimization process in a

controlled environment. This is conducive for educational purposes and for developing a robust understanding of the model's inner workings.

The architectural advantages of GPUs over CPUs, including parallel processing capabilities, faster memory access, and optimization for deep learning tasks, further underscore the rationale behind this choice, ensuring efficient and effective model training.

3. Training

The training process of this experiment was broken down into four different parts including hyperparameters and the optimizer, choice of gamma, AdaIN implementation, and execution time. These topics are elaborated on in the subsections below.

3.1 Hyper-parameters

Training a neural network efficiently requires meticulous tuning of various hyperparameters. This section outlines the specific settings utilized during the AdaIN style transfer model's training phases, ensuring transparency and reproducibility of the results.

The first hyperparameter chosen was the initial learning rate for several reasons. It was set to 1×10^{-4} . The initial learning rate is a crucial hyperparameter in the training of neural networks, playing a significant role in determining how quickly or slowly a model learns. The learning rate controls the size of the steps that the optimizer takes while updating the model's weights during training. If the initial learning rate is set too high, the model's weights might be updated too drastically, causing the model to overshoot the minimum of the loss function and potentially leading to divergence, where the model fails to converge altogether. On the other hand, if the initial learning rate is set too low, the model might take extremely small steps during training, leading to a very slow convergence process. In some cases, the model might get stuck in a local minimum and fail to reach the global minimum of the loss function. A proper learning rate ensures stable updates to the weights, preventing oscillations and instabilities in the training process. We also considered that by facilitating faster convergence to a good solution, it ends up reducing the overall training time, making the training process more efficient. When used in conjunction with learning rate decay (where the learning rate is reduced over time), setting the right initial learning rate becomes even more crucial.

Setting the correct learning rate decay was the next decision. We chose a value of 5×10^{-5} . Learning rate decay ensures that as the model gets closer to convergence, the steps it takes in the weight space become smaller. This helps in fine-tuning the weights, leading to a more precise and stable convergence. By reducing the learning rate over time, the model is less likely to oscillate around the minimum and can settle in more smoothly, which is especially important for deep and complex models. Starting with a higher learning rate allows for quicker progress initially and reducing it over time ensures stability and precision as the training progresses. This paired well with the Adam optimizer, known for its efficiency and effectiveness in training deep learning models. It adapts the learning rate during training and provides momentum through moment estimates.

When working on a local GPU with limited memory, as in the case of using a GeForce GTX 1650, adjusting the batch size is crucial to prevent out-of-memory errors. A batch size of 5 was chosen because the original parameter of 20 provided in the terminal script was too large for the GPU to handle. Smaller batch sizes introduce more variance in the gradient updates, which can sometimes help escape from sharp

minima, leading to better generalization. However, too small a batch size can also lead to instability. There is ongoing research and debate about the impact of batch size on model generalization. Some studies suggest that smaller batch sizes can lead to better generalization in certain scenarios.

For number of epochs, 20 provides a sufficient number of iterations for the model to learn and adapt its weights to capture the essential features for style transfer. It allows the model enough time to converge towards a local minimum in the loss function, which is crucial for achieving good performance. Furthermore, given the computational resources and the size of the datasets (“1K” and “10K”), 20 epochs strike a balance between ensuring adequate training and maintaining a reasonable training duration. This is particularly important to prevent excessively long training times that could hinder productivity and experimentation.

3.1.1 Gamma

We chose a value of one for gamma. Gamma acts as a balancing parameter between the content and style images in the generated output. When set to one, it ensures an equal contribution from both the content and style aspects, creating a harmonious blend in the final image.

This equal balance is often desired in style transfer applications as it allows the unique textures, colors, and patterns from the style image to be well represented, while still maintaining the overall structure and recognizable features of the content image.

Using a gamma value of one is a common practice in style transfer tasks. It serves as a standard baseline that makes the results straightforward to interpret — the contributions of style and content are equal. This simplifies the analysis of the results, especially when experimenting with various pairs of content and style images, as it provides a consistent basis for comparison across different experiments.

Furthermore, starting with a gamma value of one leaves room for easy adjustment. Depending on the specific requirements of the application or the characteristics of the images being used, increasing the influence of either the content or the style can lead to more desirable results. A gamma value of one provides a middle ground from which these adjustments can be made more intuitively. We found that after playing around with different values of gamma, keeping the value at one had the most consistent results visually.

3.2 AdaIN Implementation

The AdaIn implementation we based our code on is the AdaIn Pytorch implementation provided in the lab instructions. We include the encoder path, a pre-trained VGG network, and the decoder used in training to invert the features back to the image space [1]. Specifically, the structure of the training code, the number of layers and up sampling methods are also based on this model. Furthermore, we looked at their hyperparameter implementations to contrast and rationalize our own.

As an extension of our research, we also read through the original AdaIN paper. We were able to get a better understanding of how the neural algorithm was created and how the adaptive instance normalization layer is used at the centre of the model [2].

3.3 Execution Time

The training time for a machine learning model is influenced by various factors including the size of the dataset, complexity of the model, computational resources available, and the specific configurations used during training. In the context of our AdaIN style transfer model, the observed training times of approximately 1 hour for the 1k dataset and just under 10 hours for the 10k dataset.

The 10k dataset is roughly 10 times larger than the 1k dataset. A linear increase in the size of the dataset would theoretically result in a linear increase in the training time, assuming all other factors are constant. This is reflected in the training times reported, where the 10k dataset takes just under 10 times longer to train than the 1k dataset.

The use of a GeForce GTX 1650 GPU provides a certain level of computational power. While it is a capable GPU, it is not the most powerful available on the market. Training times could potentially be reduced with a more powerful GPU.

In addition, a smaller batch size was used to avoid running out of memory on the GPU. While this helps in terms of memory utilization, smaller batch sizes can lead to longer training times since more iterations are needed to process the entire dataset. Similarly, the number of epochs directly impacts our training time.

4. Results

The overall Results from this experiment can be analyzed through the output figure that shows the loss curves, and the 36 sample photos that have been implemented with a variety of different styles and gamma values.

4.1 Loss Curve Comparisons

After examining the training loss curves for the 1k and 10k datasets trained models, distinct differences in each model's performance are evident. The initial Content + Style Loss for the 1k dataset was notably high, starting around 13, whereas for the 10k dataset, it started lower at 6. This discrepancy indicates that the larger dataset might provide better initialization, leading to early convergence. As the training progressed, both datasets displayed a decline in losses; however, the 10k dataset exhibited a steeper descent, specifically in the Content Loss curve. The quick decline suggests an accelerated learning pace, possibly due to the diversity and abundance of examples in the larger dataset. Interestingly, the Style Loss, for both datasets, showcased a more gradual decrease, hinting that style features might be harder to capture than content features. Nonetheless, the 10k dataset consistently reported lower absolute values across epochs, ensuring its superior style learning capabilities. In conclusion, the 10k dataset not only had a faster learning rate but also resulted in better loss values, underscoring the premise in machine learning that more data typically results in enhanced model performance.

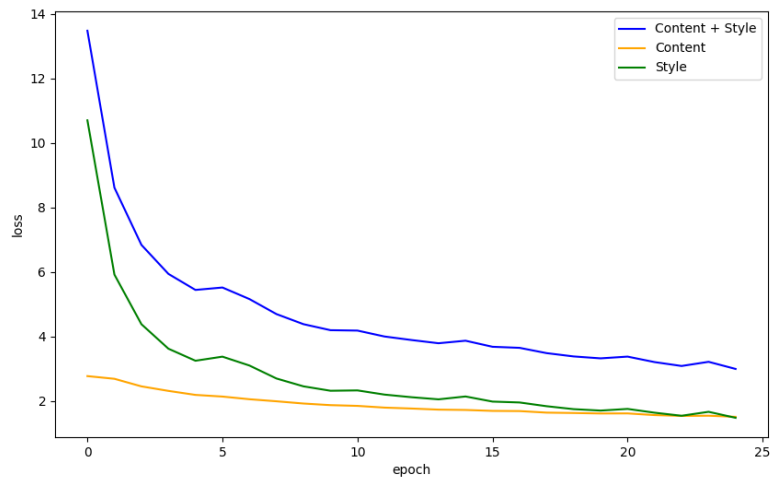


Figure 1: 1k Loss Curve

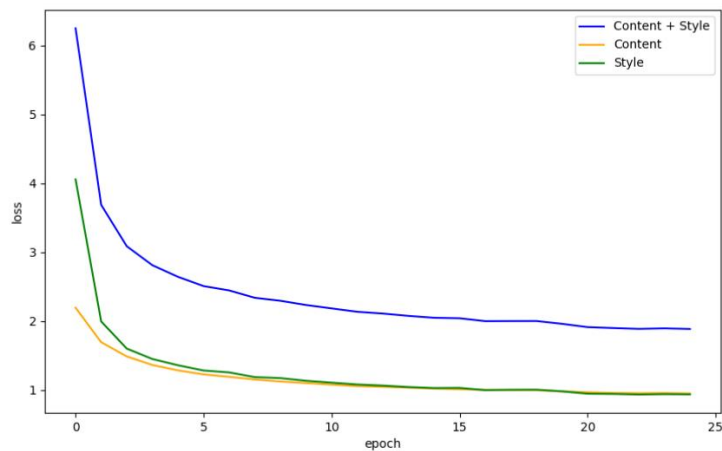


Figure 2: 10k Loss Curve

4.2 Test Comparisons

The 10K test outputs display better detail, more consistent style application, and better color preservation compared to the 1K test outputs. These qualitative differences can be attributed to the larger volume of data in the 10K model, which likely provided a richer training experience, reduced overfitting, and increased model stability. In essence, the larger dataset in the 10K tests yielded more refined and visually cohesive results than the smaller 1K dataset.

Table 1: Original Images used in Experiment.


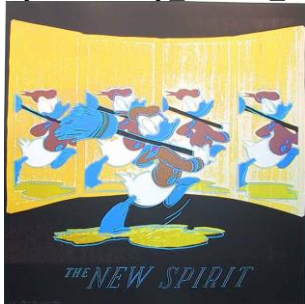
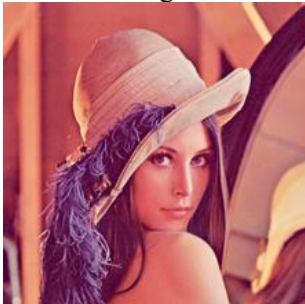

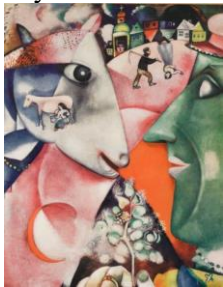
Content Pictures	Style Images
<p>Content Image 1 - ..00081 (Plane)</p> 	<p>Style 1 - Andy Warhol_97</p> 
<p>Content Image 2 – Lena</p> 	<p>Style 2 – Brushstrokes</p> 
	<p>Style 3 – ChaGall</p> 

Table 2: 81 x Andy_Warhol_97







	0.1	0.5	0.9
1k			
10K			

Table 3: 81 x Brushstrokes






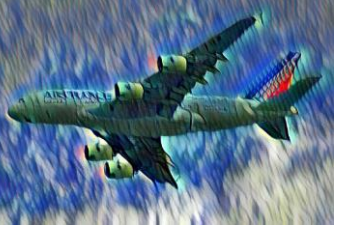
	0.1	0.5	0.9
1k			
10K			

Table 4: 81 x Chagall_Marc







	0.1	0.5	0.9
1k			
10K			

Table 5: Lena x Andy_Warhol







	0.1	0.5	0.9
1k			
10K			

Table 6: Lena x Brushstrokes

0.1	0.5	0.9
-----	-----	-----

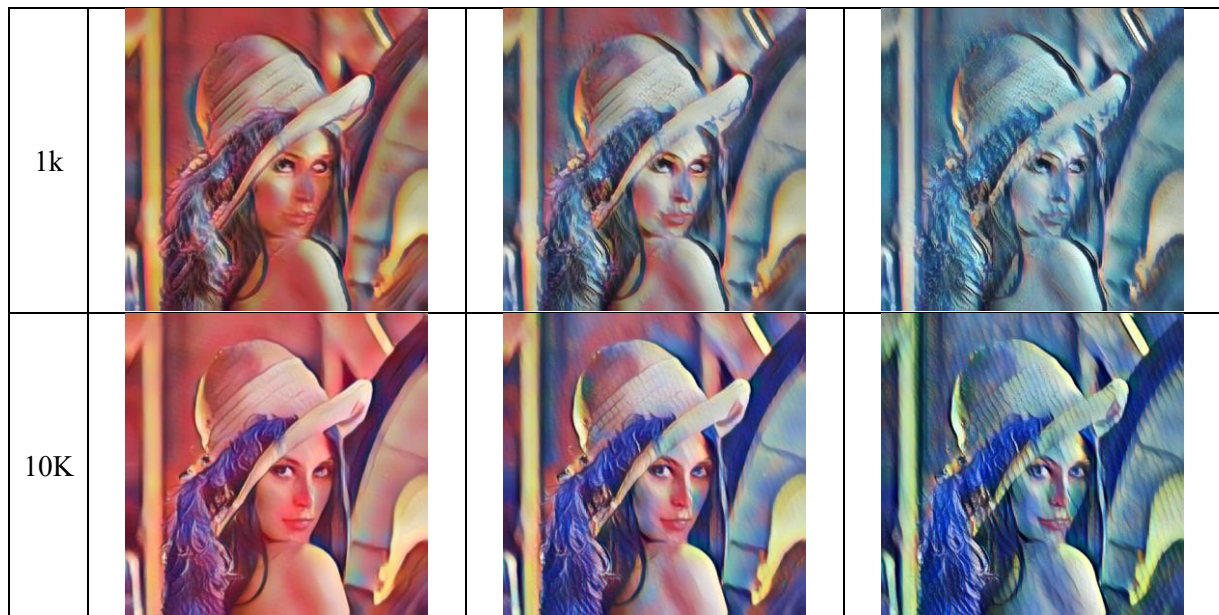


Table 7: *Lena x Chagall_Marc*



5. References

- [1] N. Inoue, “Pytorch-AdaIN: Unofficial pytorch implementation of ‘Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization’ [Huang+, ICCV2017],” *GitHub*, Oct. 04, 2023. <https://github.com/naoto0804/pytorch-AdaIN> (accessed Oct. 20, 2023).

- [2] X. Huang and S. Belongie, “Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization,” *arXiv.org*, 2017. <https://arxiv.org/abs/1703.06868> (accessed Oct. 20, 2023).