COMP9444 Project Summary

IMAGE STYLE TRANSFER

Yuan Liu (z5484761) Zhelong Zhang(z5393406)

Jiyuan Wang (z5436186) Tengjun Ni(z5490621) Jingyi Zhang (z5135098)

I. Introduction

The aim of this project is to enhance the performance and efficiency of image style transfer techniques by leveraging advanced neural network architectures. Image style transfer, which modifies an image to reflect the artistic style of another image while preserving the original content, has significant applications in digital art, advertising, entertainment, and various other creative fields.

To address the limitations of traditional convolutional neural networks (CNNs), such as handling global image information and maintaining style consistency, we develop an optimized transformer-based model and incorporate Generative Adversarial Networks (GANs). Our contributions include reducing training time through down-sampling and up-sampling techniques and enhancing image quality with GANs, ensuring higher visual fidelity and style adherence. These innovations aim to make image style transfer more effective and accessible for practical applications.

II. Related Work

We spotlight the paper "Image Style Transfer with Transformers" by Yingying Deng's team.[1] It published in 2022, introduces a novel transformer-based framework for style transfer. This approach aims to better handle the global information of images, a common shortcoming of CNN-based methods. The use of a content transformer encoder and a style transformer encoder helps capture domain-specific long-range information, while a transformer decoder translates this content into stylized images. A significant innovation here is the content-aware positional encoding which adapts to different image scales, making it particularly effective for style transfer tasks across various resolutions. However, the framework is less efficient at test-time compared to some CNN-based methods, highlighting an area for potential improvement in speed optimization.

We also referenced paper "A Dynamic ResBlock Generative 'Adversarial Network for Artistic Style Transfer" by Wenju Xu's team, which was published in 2021.[2] The paper proposes a method incorporating Dynamic ResBlocks within the GAN architecture to enhance the style and content integration during the style transfer process. This method introduces style codes as shared parameters across dynamic blocks, enabling both arbitrary and collection style transfers. DRB-GAN has shown to outperform traditional methods in terms of visual quality and efficiency. However, it still faces challenges in scaling up for high-resolution outputs and maintaining fine detail without introducing artifacts.

Both papers signify critical shifts toward using sophisticated architectures that can handle the nuances of artistic style transfer more effectively than traditional CNNs. However, both also highlight ongoing challenges such as optimizing computational efficiency and enhancing the model's ability to handle diverse artistic styles without loss of detail.

III. Methods

The project is based on StyTr2, which is a transformer-based model to achieve image transfer task.[1] There're two methods to optimize the model, one of our way to optimize the model is decline the training time of the model by down-sampling and up-sampling the data. The other way is using Generative Adversarial Network, to make the output more accurate.

For the first method, as figure 1, we added a down sample operation before the encoder entered, and used an up-sample operation to restore the previous picture state before it was passed to the decoder. With the improvement, we have increased the processing speed of the original model while basically guaranteeing the effectiveness of the image conversion task.

While using GAN, our model architecture consists of two primary components: the generator and the discriminator. As figure 2, the generator is based on the StyTr2 model, designed to produce high-quality synthetic images, while the discriminator is tasked with distinguishing these synthetic images from real images. The training process involves a competitive game between these two components, enhancing their capabilities over time.

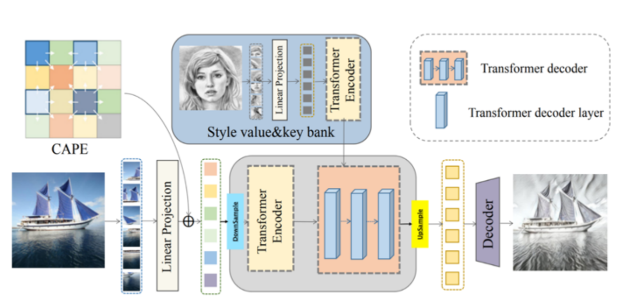


Figure 1

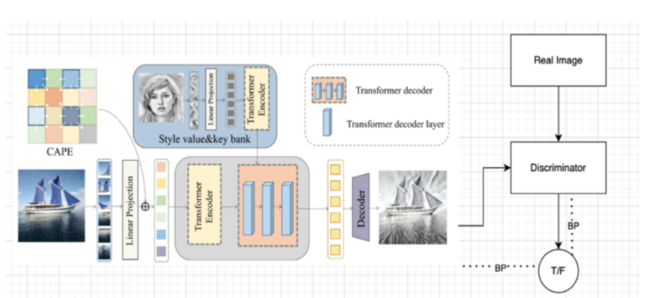
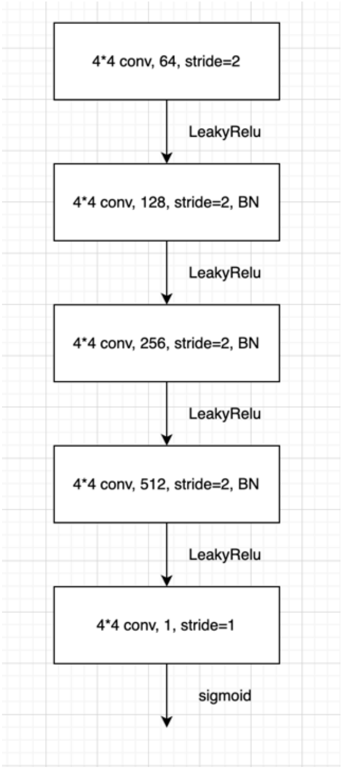


Figure 2

The discriminator's role is to evaluate the authenticity of the images by distinguishing between real and synthetic images. The architecture of the discriminator as figure:

The discriminator consists of multiple convolutional layers, including four 4x4 convolutional layers with 64 to 512 filters, strides of 2, and batch normalization (BN), followed by a LeakyReLU activation function and the last layer as 4x4 convolutional layer with 1 filter and a stride of 1, followed by a sigmoid activation function.

The training process involves adversarial training, where the generator and discriminator are trained simultaneously. The generator aims to create realistic images that can deceive the discriminator. The loss function for the generator is designed to maximize the discriminator's error in identifying synthetic images. The discriminator's goal is to accurately classify images as real or synthetic. The discriminator is trained to minimize its classification error, enhancing its ability to distinguish between real and fake images. With the gaming between the generator and discriminator and the backpropagation, the generator and discriminator progressively improve, resulting in the generation of high-fidelity synthetic images and a robust discriminator capable of distinguishing them from real images.



IV. Experimental Setup

Due to the large size of the initial dataset [3], we have decided to focus primarily on style transfer for portrait images. We downloaded dataset to the local, and then the subsequent screening process of face images. The dataset initially contained 82,784 images. The first phase of filtering was conducted using the Haar feature cascade classifier, a machine-learning based object detection method that integrates the adaboost algorithm with Haar features for quick and precise detection. This step significantly reduced the dataset to 18,181 images. For the second phase of filtering, YOLOv3 was employed, known for its real-time object detection capabilities, which further refined the dataset to 12,764 images.

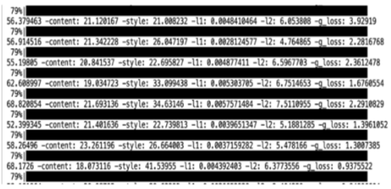
To verify the proportion of face images within our dataset, we utilized MobileNetV2, ensuring precise identification and classification of facial features. MobileNetV2 was chosen as the face recognition model mainly because of its excellent performance in terms of efficiency, accuracy and adaptability. Its deeply separable convolution and inverse residual block design enable it to provide high-quality detection results in resource-limited environments. After two rounds of data filtering, we achieved an impressive accuracy ratio of 88.03% (11,235 out of 12,764), which is essential for ensuring that subsequent analyses and models built on this data will provide meaningful and reliable insights. In order to ensure the robustness of our dataset and to avoid the pitfalls of overfitting, we have deliberately chosen not to delete the noise present in the data.

We selected the style images from the WikiArt website [4] based on two main criteria. Firstly, we sorted by count on the website and chose styles that had a larger number of images available. Secondly, since our training dataset primarily focuses on images with human faces, we opted for styles that included many portraits. These criteria helped us ensure that we have a diverse and representative set of styles for our project. Ultimately, we selected four styles: Biedermeier, Contemporary Realism, Impressionism and Ink and wash painting. Then, we wrote a script, using web scraping method to download the style datasets to our Google Drive for convenient access during training. We stored 1,000 images for each style.

V. Results

For the first method, as the figure below, the baseline training speed is about 3.6it/s. After our improvements, the training speed of the new model is 4.1it/s. For inference time, ours are also a little quicker than the method using by thesis. Our average inference speed is 0.462s, The average speed of reasoning in papers is 0.661.

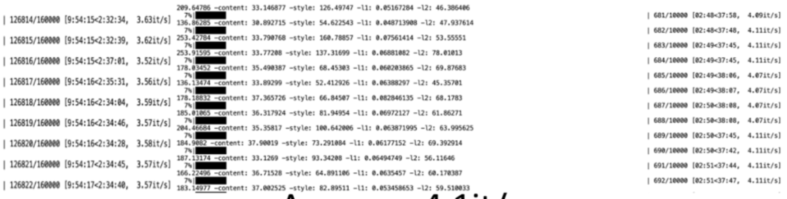
Original Train Time



```
 79%|████████████████████████████████████████
56.379463 -content: 21.120167 -style: 21.008232 -l1: 0.0048410464 -l2: 6.053808 -g_loss: 3.92919
 79%|████████████████████████████████████████
56.914516 -content: 21.342228 -style: 26.047197 -l1: 0.0028124577 -l2: 4.764865 -g_loss: 2.2816768
 79%|████████████████████████████████████████
55.19805 -content: 20.841537 -style: 22.695827 -l1: 0.004877411 -l2: 6.5967703 -g_loss: 2.3612478
 79%|████████████████████████████████████████
62.608997 -content: 19.034723 -style: 33.099438 -l1: 0.005303705 -l2: 6.7514653 -g_loss: 1.6760554
 79%|████████████████████████████████████████
68.820854 -content: 21.693136 -style: 34.63146 -l1: 0.0057571484 -l2: 7.5110955 -g_loss: 2.2910829
 79%|████████████████████████████████████████
52.399345 -content: 21.401636 -style: 22.739813 -l1: 0.0039651347 -l2: 5.1881285 -g_loss: 1.3961052
 79%|████████████████████████████████████████
58.26496 -content: 23.261196 -style: 26.664003 -l1: 0.0037159282 -l2: 5.478166 -g_loss: 1.3007385
 79%|████████████████████████████████████████
68.1726 -content: 18.073116 -style: 41.53955 -l1: 0.004392403 -l2: 6.3773556 -g_loss: 0.9375522
 79%|████████████████████████████████████████
```

Average: 3.60it/s

Our Train Time

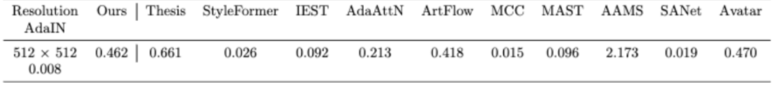


```
                                            209.64786 -content: 33.146877 -style: 126.49747 -l1: 0.05167284 -l2: 46.386406
                                              7%|████
| 126814/160000 [9:54:15<2:32:34,  3.63it/s] 136.86285 -content: 30.892715 -style: 54.622543 -l1: 0.048713908 -l2: 47.937614
                                              7%|████
| 126815/160000 [9:54:15<2:32:39,  3.62it/s] 253.42784 -content: 33.790768 -style: 160.78857 -l1: 0.07561414 -l2: 53.55551
                                              7%|████
                                            253.91595 -content: 33.77208 -style: 137.31699 -l1: 0.06881082 -l2: 78.01013
| 126816/160000 [9:54:15<2:37:01,  3.52it/s]  7%|████
                                            178.03452 -content: 35.490387 -style: 68.45303 -l1: 0.060203865 -l2: 69.87683
| 126817/160000 [9:54:16<2:35:31,  3.56it/s]  7%|████
                                            136.13474 -content: 33.89299 -style: 52.412926 -l1: 0.06388297 -l2: 45.35701
| 126818/160000 [9:54:16<2:34:04,  3.59it/s]  7%|████
                                            178.18832 -content: 37.365726 -style: 66.84507 -l1: 0.082846135 -l2: 68.1783
                                              7%|████
| 126819/160000 [9:54:16<2:34:46,  3.57it/s] 185.01065 -content: 36.317924 -style: 81.94954 -l1: 0.06972127 -l2: 61.86271
                                              7%|████
                                            204.46684 -content: 35.35817 -style: 100.642006 -l1: 0.063871995 -l2: 63.995625
| 126820/160000 [9:54:16<2:34:28,  3.58it/s]  7%|████
                                            184.9082 -content: 37.90019 -style: 73.291084 -l1: 0.06177152 -l2: 69.392914
| 126821/160000 [9:54:17<2:34:45,  3.57it/s]  7%|████
                                            187.13174 -content: 33.1269 -style: 93.34208 -l1: 0.06494749 -l2: 56.11646
                                              7%|████
| 126822/160000 [9:54:17<2:34:40,  3.57it/s] 166.22496 -content: 36.71528 -style: 64.891106 -l1: 0.0635457 -l2: 60.170387
                                              7%|████
                                            183.14977 -content: 37.002525 -style: 82.89511 -l1: 0.053458653 -l2: 59.510033
```

```
| 681/10000 [02:48<37:58,  4.09it/s]
| 682/10000 [02:48<37:48,  4.11it/s]
| 683/10000 [02:49<37:45,  4.11it/s]
| 684/10000 [02:49<37:45,  4.11it/s]
| 685/10000 [02:49<38:06,  4.07it/s]
| 686/10000 [02:49<38:07,  4.07it/s]
| 687/10000 [02:50<38:08,  4.07it/s]
| 688/10000 [02:50<38:08,  4.07it/s]
| 689/10000 [02:50<37:45,  4.11it/s]
| 690/10000 [02:50<37:42,  4.11it/s]
| 691/10000 [02:51<37:44,  4.11it/s]
| 692/10000 [02:51<37:47,  4.11it/s]
```

Average: 4.1it/s

Inference Time



Resolution AdaIN	Ours	Thesis	StyleFormer	IEST	AdaAttN	ArtFlow	MCC	MAST	AAMS	SANet	Avatar
512 × 512 0.008	0.462	0.661	0.026	0.092	0.213	0.418	0.015	0.096	2.173	0.019	0.470

For the second method, as the figure below, the performance of our model can be observed through the generator loss (G_loss) and the discriminator loss (D_loss). It is evident that the discriminator exhibits stronger performance compared to the generator. This imbalance indicates that the generator has not yet achieved sufficient robustness to effectively challenge the discriminator. However, this disparity falls within an acceptable range. In the adversarial training process, it is typical for the generator loss to fluctuate periodically, initially increasing and then decreasing as the generator improves.

However, the results are worse than the baseline, especially for content loss. The possible reason why our model is not better than the baseline is that, for the real data given to the Discriminator is weak cause we only use the style picture as the real data, while the data is not as large and robust as content picture data. In the future, we need more style data and we need decrease noises from the vague style pictures. Some of those style pictures are too old so there're lots of noises on it.

Metric	Ours	Baseline
Loss_content	19.605549	17.7026602
Loss_style	27.4924179	27.1744588
Loss_identity1	0.00450177	0.00386792
Loss_identity2	5.73630982	4.15021229
Total_loss	58.0097836	49.2980859
G_loss	2.43019144	
D_loss	0.3042345	

VI. Conclusions

For this project, our group made the following main contributions. The first is the improvement of the environment. Since the environment and version provided by the original project were buggy, we set up a runnable and compatible training environment of Ubuntu 22.04 on our server.

After that, our data preprocessing firstly used a crawler to download the style images we needed, and then used yolov3 and MobileNetV2 to filter the portrait images we needed, which ensured that the final training data reached a correct rate of 88%. Finally, the improvement method we used.

Our first improvement method is to ensure improved inference and training speed by using downsample before entering the encoder and upsample before entering the decoder. Due to the limitation of the number of interfaces in the decoder, we can only upsample before decoder. this may cause our overall speedup still has a lot of room to be improved by modifying the decoder. also due to downsampling, although we chose portrait processing to avoid the interference of the large details, there may still be missing.

The second method is to add a discriminator on top of the original network to form a Generative adversarial network, which is based on the principle of adding a discriminator consisting of multiple convolutional layers. The generator and the discriminator are trained at the same time, and through the game and backpropagation between the generator and the discriminator, the generator and the discriminator are gradually improved, so that the discriminator can better identify the errors in the synthetic images, and the generator can generate more similar images in the adversarial process, so that their abilities are eventually improved. Through this method, we can make GAN better able to capture the distributional features of the data through adversarial training, to generate more realistic and natural data. However, due to the low input of real style images and low-quality images for our discriminators, and the overabundance of information in the transformed images compared to the others, it will lead to a high loss of our content. Afterwards, we will improve our results by further processing of the trained style images, such as reducing image noise.

Reference

[1] Deng Y, Tang F, Dong W, et al. Stytr2: Image style transfer with transformers[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022: 11326-11336.

[2] W. Xu, C. Long, R. Wang, and G. Wang, "Drb-gan: A dynamic resblock generative adversarial network for artistic style transfer," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 6383-6392. Available at:https://openaccess.thecvf.com/content/ICCV2021/papers/Xu_DRB-GAN_A_Dynamic_ResBlock_Generative_Adversarial_Network_for_Artistic_Style_ICCV_2021_paper.pdf

[3] J. Faudi, "COCO 2014 Dataset (for YOLOv3)," Kaggle.com, 2014. https://www.kaggle.com/datasets/jeffaudi/coco-2014-dataset-for-yolov3

[4] "WikiArt.org - Visual Art Encyclopedia," www.wikiart.org. https://www.wikiart.org