INFO8010: Style Transfer

"Altaha Yaman"[1]

[1] **yaman.altaha@student.uliege.be** *(s215427)*

I. INTRODUCTION

In this project we used convolutional neural networks to achieve artistic style transfer on images. CNN's have demonstrated their strength in object recognition, detecting objects and discerning image content. The system uses neural representations to separate and recombine the content and style of arbitrary images, to achieve artistic style transfer by providing artistic image creation. Since this project has already been done, several resources helped us in our task such as Neural Algorithm of Artistic Style [1] which gave us a clear approach to solve this problem and some resources will be mentioned later. We used some pretrained models (VGG19, ResNet50), as well as a VGG19 network that we implemented ourselves. The main challenge is to create a network that takes both the content image and the style image as input and generates an output image that merges the content of the former with the style of the latter and gives valid results.

II. RELATED WORK

Thanks to Convolutional Neural Networks (CNN), deep learning networks achieve some pretty impressive results in the world of computer vision . In fact, CNN is really useful for classification, object detection, or even semantic segmentation. With convolutional neural networks trained on a classification task, we could separate the content and style representations of images and then combine them to create stylized images. We based our work on the original paper for this style transfer method [1], which used a VGG19 neural network. After we failed to reproduce the results using a ResNet50 network, we also quickly reviewed a more recent paper [2] about the same technique applied to more modern CNNs that make use of residual connections to improve gradient flow. It turns out that the residual connections themselves lead to bad results for style transfer, although some mitigations are possible.

In the context of our project focus on style transfer, it is useful to explore existing research and studies that focus on similar areas. By reviewing these relevant papers, we can get more knowledge, ideas and learn more about our work that can guide our efforts to better ways.

Let's consider the paper **Perceptual Losses for Real-Time Style Transfer and Super-Resolution** [3] which describes a method for transferring artistic styles using perceptual losses. With deep neural networks (CNNs), it achieves real-time style transfer and captures high-level content. The main idea in this paper is to compute perceptual losses using a pre-trained deep convolutional neural network (VGG-16), which are then used to improve the generated image during style transfer.

However, this raises the question of how the perceptual losses are computed, instead of directly using pixel-wise differences between the generated image and the target style. in this paper they used feature representations obtained from pre-trained VGG network to compute the perceptual loss. The perceptual loss determines the difference in feature representations between the generated image and the style image. The overall objective function minimizes the perceptual loss alongside the content loss, ensuring that the transformed image preserves its content while acquiring the style of the reference image.

Figure 1 illustrates the system architecture in the paper. It consists of two main components: an image transformation network denoted as f_W, which converts input images x into output images $\hat{y}$ through the transformation $\hat{y} = f_W(x)$, and a separate loss network ϕ which defines multiple loss functions. These functions measure the perceptual difference between generated and target images. The image transformation network is trained to minimize a combination of these loss functions, guided by the fixed loss network.

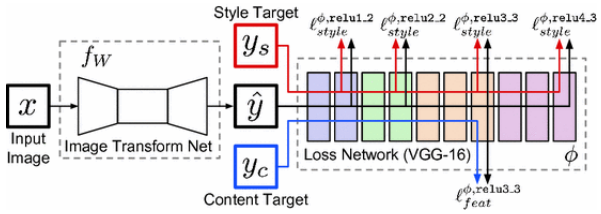

FIG. 1. Overview of the system

According to the study, they confirmed the effectiveness of using perceptual losses for artistic style transfer, which opens up new possibilities for real-time style transfer applications. Additionally, in terms of the architecture, it's built with a feed-forward neural network which facilitates real-time style transfer. However, they also show how their method can be extended to handle super-resolution tasks while still keeping image quality improvement. In the end, we may conclude from their results that they achieved great results in terms of visual quality as well as computational efficiency.

A Neural Algorithm of Artistic Style [1] and

Perceptual Losses for Real-Time Style Transfer and Super-Resolution[3] both make great advances in the field of artistic style transfer. However, there are some differences in terms of optimization methodologies, performance, and additional applications. While the first study introduced the basic concept of neural style transfer, the latter study further advanced the field by allowing real-time style transfer and extending to handle with super-resolution problems.

As an example of another important contribution is the paper titled **Domain-Aware Universal Style Transfer**, which introduces a unified architecture that addresses the limitations of both artistic and realistic style transfer models. While artistic style transfer models are successful in capturing artistic features, they fail to preserve the structural elements of photo-realistic images. On the other hand, photo-realistic style transfer models do a better job of preserving structure but struggle to properly capture artistic features effectively.

The proposed model in this paper overcomes these challenges through the use of domain-aware skip connections, which are crucial in differentiating between artistic and realistic images while preserving structural elements. These skip connections are the key difference between existing artistic and photo-realistic models. However, the domain-aware style transfer network (DSTN) consists of an auto-encoder equipped with domain-aware skip connections and a "domainness" indicator.

The "domainness" indicator is designed to determine whether an image falls under the artistic or photo-realistic domain, or a mix of both, and it is critical for the adaptive transfer of stroke and palette from the reference image to the input content. The domainness indicator is determined based on both structural and textural features, using feature maps from the encoder and texture information computed using a gram matrix $G(f_l)$ of feature maps f_l, and is represented by a parameter that can take one of three values (classifying the image into one of three domains): photo-realistic, artistic, or mixed. The mixed domain is trained using binary cross-entropy and is designed to reside between the two domains allowing for smooth transitions between styles.

Figure 2 shows an overview of the domain-aware style transfer networks (DSTN) method. In part (a), the process begins with the proposed auto-encoder featuring domain-aware skip connections for multi-domain style transfer. Part (b) shows how to extract high-frequency components from a stylized feature. Part (c) goes into further detail about feature transformation.

To evaluate the performance of DSTN, the domainness parameter are trained by using binary cross-entropy, and the model is trained on COCO and WikiART datasets, each containing over 80,000 images, representing a wide range of artistic and photo-realistic styles.

The results show that DSTN provides advanced performance both quantitatively and qualitatively without the need for any preprocessing or post-processing steps, achieving impressive performance in both artistic and photo-realistic domains. While artistic style transfer models achieve acceptable style loss scores, they often fail to preserve the original structure of the image. On the other hand, photo-realistic style transfer models have the opposite issue.

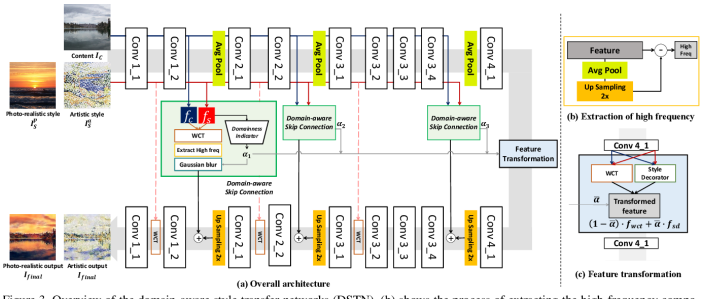

FIG. 2. Overview of DSTN system

Finally, "Domain-Aware Universal Style Transfer" offers a comprehensive solution for successful style transfer in both artistic and photo-realistic domains. The model effectively preserves structural elements and captures the domain characteristics of images, while maintaining artistic features by incorporating domain-aware skip connections and the domainness indicator, making it an outstanding approach in the field domain-aware universal style transfer.

III. METHODOLOGY

The following steps are involved:

A. Style Transfer Implementation

1. *Preprocessing*

Preprocessing is done in the way that was used for training the networks.

- Resize the images to a fixed size for further processing.
- Normalize the pixel values for both the content and style images to ensure they are in a similar range.

2. *Loss Functions*

We define a loss function to quantify the divergence in style and content between two images, using the trained CNN.

- Content loss:

  Calculate the squared difference between the feature representations of the content image and the generated image.

  - Extract the feature maps of the content image and the generated image at a specified layer of the pre-trained CNN.
  - Compute the total squared error between the feature maps to measure the content variation.

- Style loss:

  Capture the style of the style image by comparing the Gram matrices of their feature representations. The Gram matrices encodes the correlations between the activations of the different filters of a layer.

  - Extract the feature maps of the style image and the generated image at multiple layers of the CNN.
  - Calculate the Gram matrices of the feature maps, which represent the correlations between their activations.
  - Compute the total squared error (normalized for the layer sizes) between the Gram matrices to measure the style difference.

- Total loss:

  The total loss is the weighted sum of the style and content losses. The best results were achieved with a style loss of weight 1 and content loss of weight 10^{-2}.

3. *Training Parameters*

With the following parameters were used with Adam optimizer shown in the table I, we achieved fairly acceptable style transfer outcomes. These parameter choices allowed the content and style to be integrated into the generated results.

Content weight	0.01
Style weight	5
Learning rate	0.02
Number steps	5000

TABLE I. Training parameters

4. *Optimize the generated image*

- Initialize a random noise image or a copy of the content image as the generated image. Starting with a copy of the content image make the optimization process much faster and reduces the need for the content loss. In some cases, it can almost be removed.

- Define an optimization process to minimize the total loss function. We used the Adam optimizer with default parameters and a learning rate of 0.02, doing 5000 iterations.

- Use backpropagation to compute the gradients of the loss with respect to the generated image.

- Update the generated image by taking a step in the direction of the negative gradients.

  Finally, repeat the optimization process for a certain number of iterations to improve the generated image.

B. Convolutional neural network implementation

1. *Custom VGG19-based Architecture*

The network architecture used is based on VGG19 as shown in Figure 3. It includes 16 convolutional layers with ReLU activations divided into 5 convolutional blocks. 4 max-pooling layer for spatial reduction. Furthermore, 1 adaptive average pooling is used to reshape the feature maps into a fixed size. In addition, there are three fully connected layers with ReLU activations and dropout to prevent overfitting.

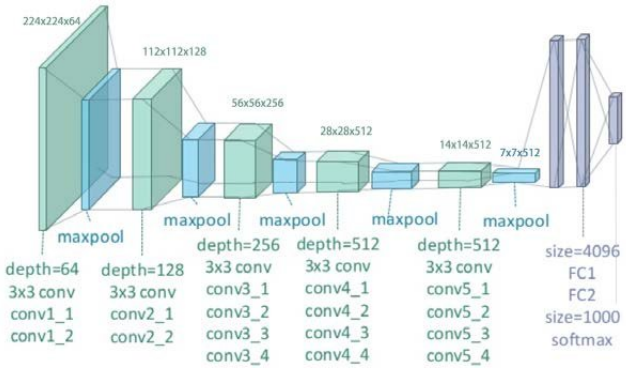

FIG. 3. Illustration of the network architecture of VGG-19 model

In addition, here are some improvements and differences between our initial architecture and the upgraded version to improve the network architecture:

- Define the VGG19 model by subclassing nn.Sequential, which is a cleaner and readable way to define a deep model.

- Instead of include a separate flatten layer, an adaptive average pooling layer was added after the con-

volutional layers and before the fully connected layers to ensure that the subsequent fully connected layers receive input of a fixed size.

- In terms of weight initialization, our previous architecture was developed without explicitly include weight initialization.This meant that the network's parameters were initialized using the default settings. Since the architecture mostly uses ReLU activations, it was found that the use of Xavier Initialization is not a good choice. However, using Kaiming Initialization is a better solution because of its compatibility designed with heavy use of ReLU activations in the architecture being used.

- Add a forward pass in which input data is passed through the neural network's layers to produce predictions. It is essential because it allows the network to understand the input data and make accurate predictions based on learned features.

In general, deeper networks perform better. Especially when the network architecture is more complex and deeper, it often performs better but requires more computational resources and longer training time.

2. Loss function

Cross-entropy loss function has been chosen because it works well with multi-class classification tasks and guides the model during training to improve its classification accuracy.

3. Dataset

The CIFAR-100 dataset has been chosen for the new implementation, it has 100 different types of images, with 600 images in each type, and it is considered quite large, with 60,000 images in total. This bigger collection helps the new model to learn more effectively from the Caltech-101 dataset that was used earlier.

IV. RESULTS

This section presents all the outcomes of our experiments and shows how the different methods we tried affected our results, including how fast our model learned and how accurate it became. We tested our method using various networks: pretrained VGG19, pretrained ResNet50, our own VGG19 network, Recently, in addition the new version of VGG19 after adding improvements to it.

A. Experimental Results

Previously, we trained on the Caltech101 dataset, due to the small dataset and because we ran out of time at that point, so we trained only with 5 epochs. However, since the VGG19 is a very large model, that training was ineffective. In Figure 4 we can see that the results were not good as the test accuracy, as well as the training and test loss showed unsatisfactory values. This shows that our short training and the small dataset made things difficult for our previous model.

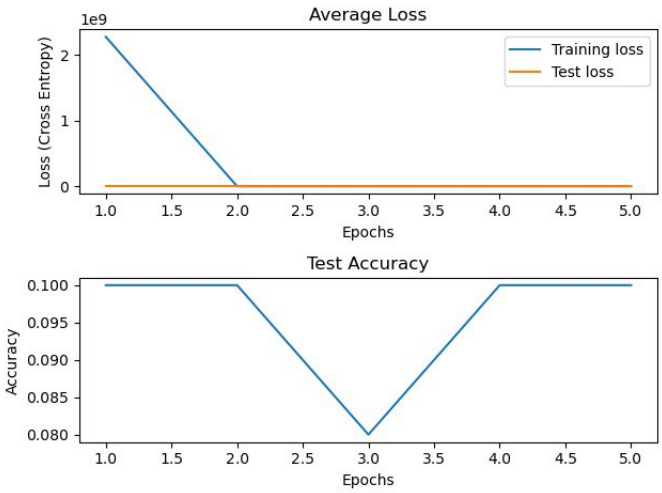

FIG. 4. Loss and test error of our previous VGG19 network during training on Caltech101

However, in an attempt to improve our previous model, the number of epochs has been increased to 150 and set the learning rate to 0.001, hoping to obtain better results than before, but as shown in the figure 5, the results are still unsatisfactory and the accuracy is basically didn't move.

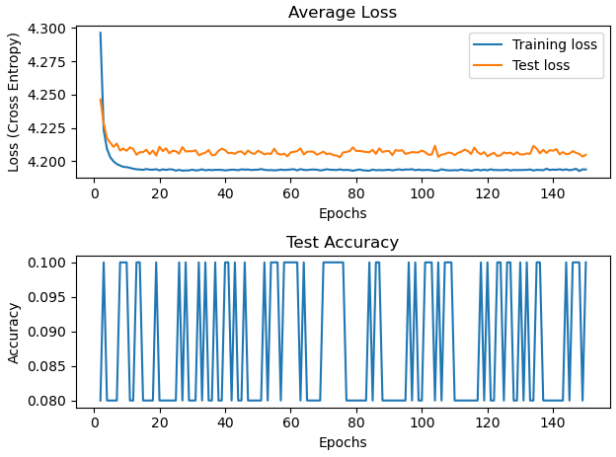

FIG. 5. Loss and test error of our previous VGG19 network during training on Caltech101

Additionally, We also tried using ResNet50 but it gave quite poor results despite selecting the same layers as in [2]. Figure 6 shows the instability, and links it to the poor results and disappointing ResNet50 performance.

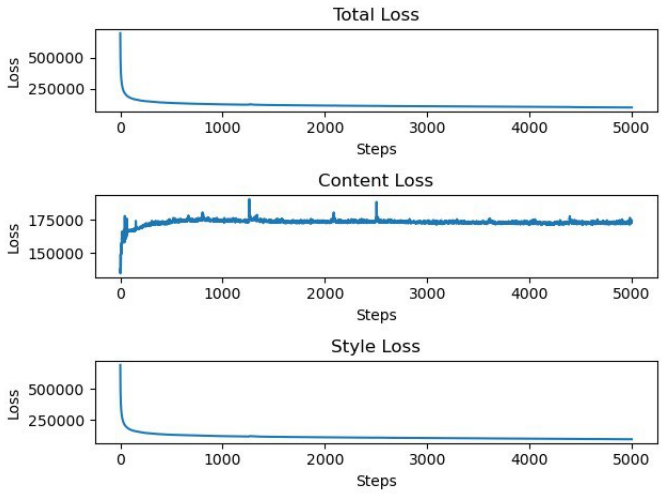

FIG. 6. Loss during optimization of generated image with pretrained ResNet50

B. Enhancements for Improved Outcomes

Eventually, after using the new deeper network and making optimizations and enhancements that were implemented, training efficiency, accuracy, and the performance resulted in a significant improvement along with training speed and memory efficiency.

Many improvements have been made, including the following:

- Use mixed precision training technique [4] found in Pytorch documentation, it combines float32 and float16 precision via autocast to speed up training, reduce memory usage and computation time while maintaining accuracy.
- Make some modifications within the training loop to make it run faster in each iteration.
- Reduce memory usage in terms of data loading
- Use AdamW [5] instead of Adam which is better for many reasons, since it includes weight control, enhancing model stability, reducing overfitting and faster training, which leads to better performance.
- Split the dataset into training and testing sets using the trainset and testset from torchvision's CIFAR-100 dataset, instead of using random split.
- Add more data preprocessing steps such as different resizing, and augmentation techniques, which greatly affect the testing accuracy.
- Use ReduceLROnPlateau [6] learning rate scheduler, it automatically adjusts the learning rate during training, which helps to improve training efficiency and accuracy.

As a result, these improvements and modifications led to faster performance, and significantly increased testing accuracy, allowing the model to be trained more efficiently.

C. Actual Results

In the beginning, the initial improvements made, when 20 epochs took about 6 hours to complete, with a testing accuracy of about 52%, as shown in the figure 7.

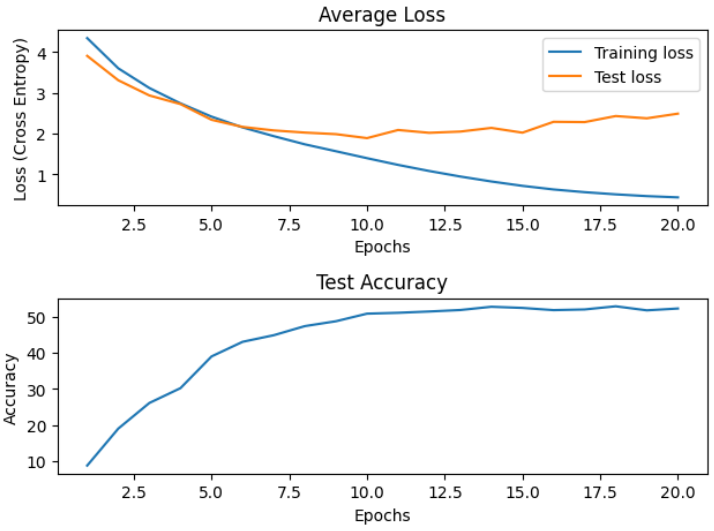

FIG. 7. The new VGG19 network during training on CIFAR-100 dataset

After implementing all the improvements that were previously mentioned, 20 epochs only took about two and a half hours, which is about halfway the time it used to take, while maintaining almost the same good testing accuracy rate of about 50%, which is a very slight small difference from the previous one, and this is considered good. Figure 8 shows how little the accuracy has changed since before.

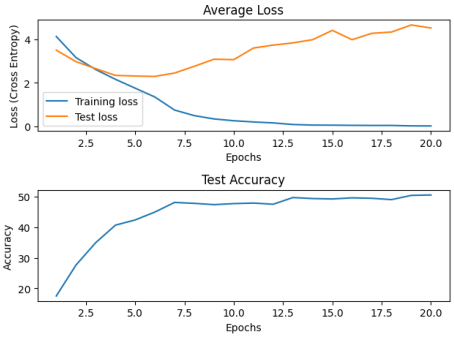

FIG. 8. The new VGG19 network during training on CIFAR-100 dataset

Therefore, after ensuring that the model somehow works well and the accuracy is acceptable (while noting that there is a possibility of overfitting) . Since the model works faster than it was previously, this allowed the possibility of setting the number of the epochs to a rather large number. However, training neural networks in general always take a long time. Figure 9 shows the final test accuracy result obtained after 120 epochs, which is about 50% and required a long time to complete. Knowing that it needs to be addressed overfitting to enhance the model generalization capabilities and improve its performance on new and unseen data.

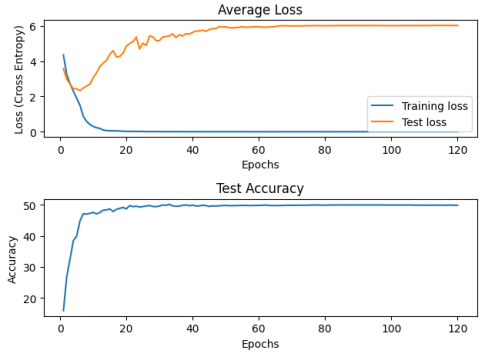

FIG. 9. The new VGG19 network during training on CIFAR-100 dataset

After training the updated version of VGG19 to extract feature maps and then running the style transfer, unfortunately the results were not as should be despite having an accuracy of about 50%, the outcomes were unsatisfactory.
Figure 10 show the results. There could be many reasons, perhaps the the need to go through many more epochs in the training phase, but they require a very long time, or the dataset should be changed to a larger one that includes more image categories.

FIG. 10. Style transfer result using the new trained version of VGG19

As we can see from Figure 11 that the only good result is obtained by using the pretrained VGG19.

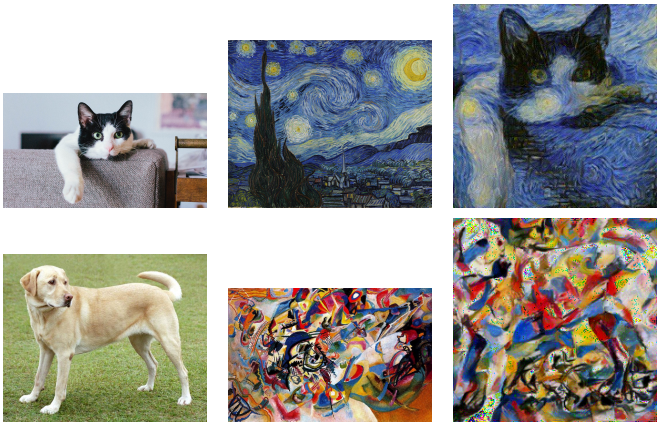

FIG. 11. Style transfer result using pretrained VGG19

Figure 12 shows the loss during optimization of the generated image with pretrained VGG19. We have to make it clear that the reason for the content loss is going up, because we initialize the gradient descent with the content image, which logically has a content loss of 0 and can thus only go up.

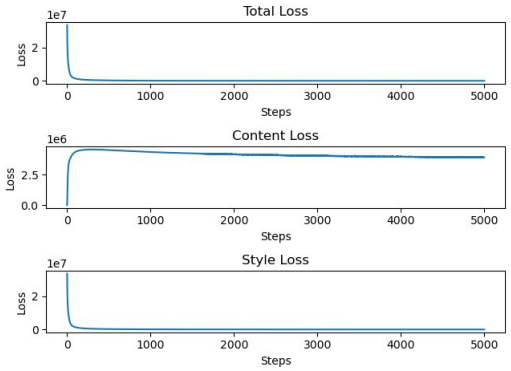

FIG. 12. Loss during optimization of generated image with pretrained VGG19

V. DISCUSSION

A. Comparing Neural Network Results and Challenges

We can say that the pretrained VGG19 model has given the best results when compared to all of the networks tested, where they are rather satisfactory and acceptable. This is due to the effectiveness of our method that are used which contains fine tuning parameters. However, it was less satisfying with the artistic style transfers generated using the new improved version of VGG19, despite achieving about 50% accuracy after the training. While the results were not as optimal as it should be, the generated image is somehow acceptable because it seems to be taking some artistic style to some extent. We might get a better image if we increase the number of epochs to a large number, however this takes a lot of time and needs efficient resources. The results consistently showed that the improved architecture worked better than the previous version in terms of accuracy, performance, outcomes and other factors.

B. Limitations and Future Work

For the future work, either place more attention on pretraining neural networks or it would more be beneficial to focus more on getting the neural network ready and train it well with better accuracy rate. From the obtained results, we may conclude that having a well-tuned Convolutional Neural Network can lead to faster learning and better outcomes.

C. Other Considerations

The implementation of "A Neural Algorithm of Artistic Style" [1] involves utilizing a pre-trained VGG network to extract content and style representations from images. By minimizing a combined loss function, the algorithm generates visually appealing images that blend content and style characteristics. The paper's approach has shown impressive results and opens up possibilities for various applications in image synthesis and artistic rendering. With the right architectures, we can get good results using well trained networks, but if either the architecture changes or the network is not well trained, the results will not be great. On the other hand, "Rethinking and Improving the Robustness of Image Style Transfer" [2] suggests ways to make style transfer more effective using networks that have residual connections like ResNet.

VI. CONCLUSION

As it was depicted above, the outcomes were satisfactory, where a substantial improvement on the final results can be clearly seen comparing to the last work.

The project was a big challenge to me. Since I was solely responsible for implementing the new improvements, it took a lot of efforts, time, consultations and research and I did my best to improve the project and meet all the new requirements that needed to be done.

[1] Matthias Bethge Leon A. Gatys, Alexander S. Ecker. A neural algorithm of artistic style, 2015.

[2] Nuno Vasconcelos Pei Wang, Yijun Li. Rethinking and improving the robustness of image style transfer, 2021.

[3] Li Fei-Fei Justin Johnson, Alexandre Alahi. Perceptual losses for real-time style transfer and super-resolution, 2016.

[4] `https://pytorch.org/docs/stable/notes/amp_examples.html`.

[5] Frank Hutter Ilya Loshchilov. Decoupled weight decay regularization, 2017.

[6] `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html`.

[7] `https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html`.

[8] `https://pytorch.org/docs/stable/nn.init.html`.

[9] `https://iq.opengenus.org/vgg19-architecture/`.

[10] `https://jaketae.github.io/study/pytorch-vgg/`.