Capstone report: Style Transfer Learning

Udacity AWS Machine Learning Engineer Nanodegree

Camila Andrea González Williamson

April 16, 2023

1 Domain Background

Style transfer is a popular application of deep learning that involves transforming an input image (e.g., a photograph or a video) into another image into the style of another image (e.g., a painting or a texture). Figure 1 illustrates an example of Style Transfer, by applying to the image of a cat (content image) the style of 3 different artworks (style images). This technique has many potential uses, such as the artistic rendering of photographs, graphic design, and film special effects. One popular approach to style transfer is to use convolutional neural networks (CNNs), which can learn to extract and manipulate visual features in an image [1]. In recent years, many researchers have explored various techniques for improving the quality and efficiency of style transfer models.

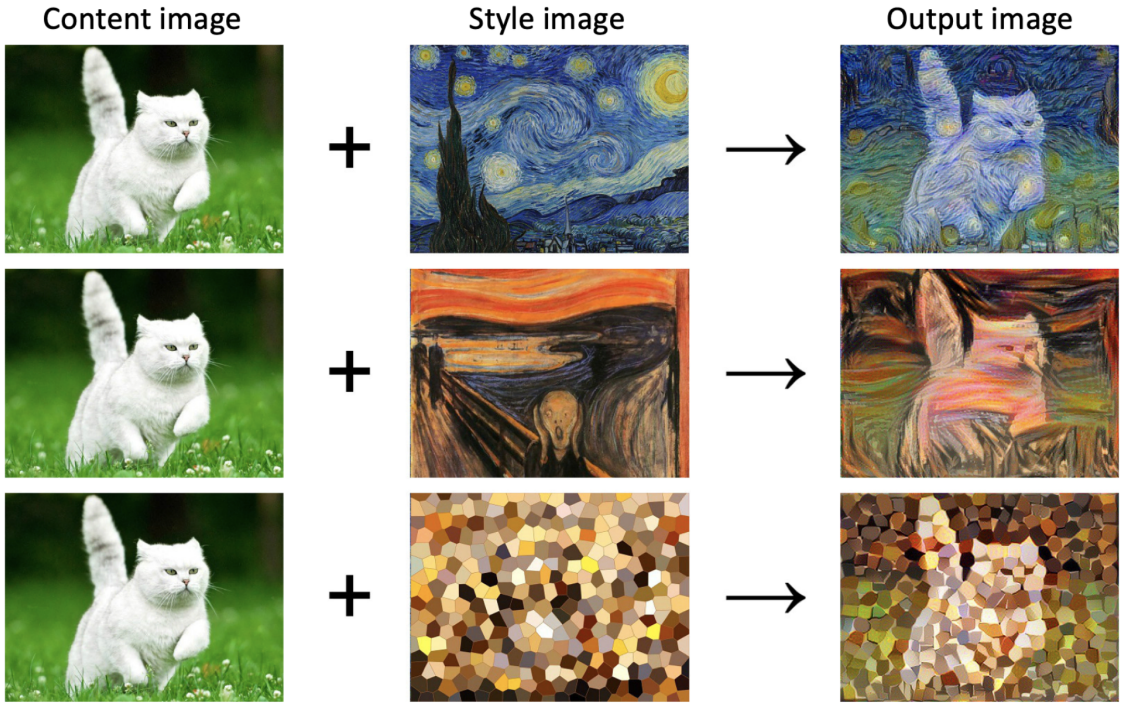

Figure 1: Example of transfer learning. Taken from [2]

2 Problem Statement

Nowadays there are various ready-made models for Style Transfer Learning like the one provided by TensorFlow Hub [3]. Most of these models use Fast Style Transfer, a technique that can generate stylized images in real-time, or near real-time, but often at the expense of image quality. Figure 2 shows an example of an image produced using Fast Style Transfer and traditional Neural Style Transfer.

Figure 2: Images produced using Fast Style Transfer (left) and traditional Neural Style Transfer (right). Adapted from [4]

The goal of this project is to implement a traditional Neural Style Transfer model in AWS Sagemaker that produces higher quality stylized images than fast style transfer models. The resulting model will take a content image, and a style image, and produce a new image by transferring the style onto the content image. The resulting stylized images are evaluated for their visual quality using appropriate metrics and compared to images produced by Fast Style Transfer. In particular, we use as evaluation metrics the Mean Squared Error (MSE) and the Structural Similarity Index (SSIM). The SSIM is of particular importance as it measures the similarity between images based on their perceptual features, luminance, contrast, and structure, rather than their pixel values. It is also used to evaluate image quality degradation caused by losses in data transmission or by processing such as data compression.

3 Datasets and Inputs

3.1 Content image

To ensure consistency in the quality evaluation of the generated image, we use a single image as a content image across all experiments. Figure **??** shows the image selected. It consists of a gray sphynx cat with a width of 512 pixels and a height of 459 pixels.

Figure 3: Selected content image: gray sphynx cat. Sourced from [5]

3.2 Style images

Typically, style images consist of painting images from known artists. To source the painting images we used the Kaggle dataset "Best Artworks of All Time", which is a collection of Paintings of the 50 Most Influential Artists of All Time, scrapped from artchallenge.ru at the end of February 2019 [6].

The dataset comprises a total of 8118 painting images. Figure 4 shows the number of painting images per artist. As we can see, the distribution is highly imbalanced across artists. There is a median of 120 images per artist with an interquartile range of 101 images (75th percentile - 50th percentile). The artist with the highest number of images is Vincent Van Gogh, with 877 images, and the artist with the lowest number of images is Jackson Pollock with only 24 images.

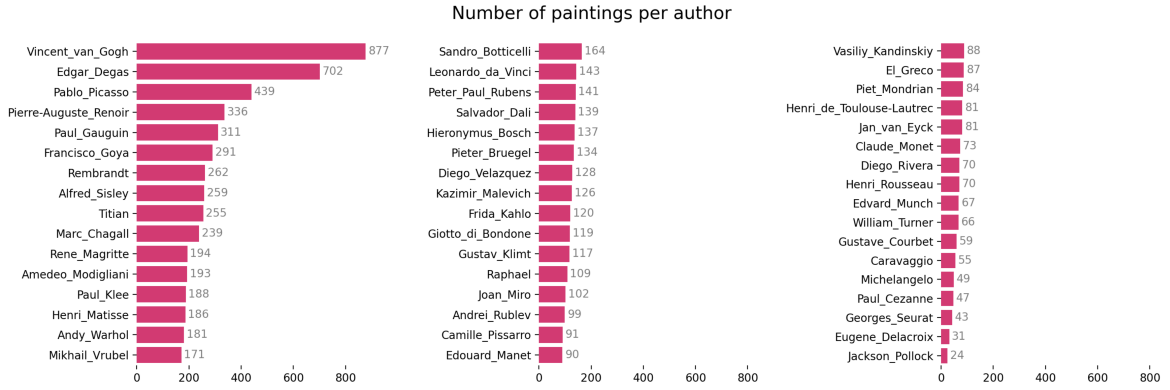

Figure 4: Distribution of painting images across artists

With respect to the size of the images, the median width is 807 pixels (with a min. of 204px and a max of 4096px) and the median height is 923 pixels (with a min, of 226px and a max. of 3826px). Figure 5 shows the median width and height of painting images per artist.

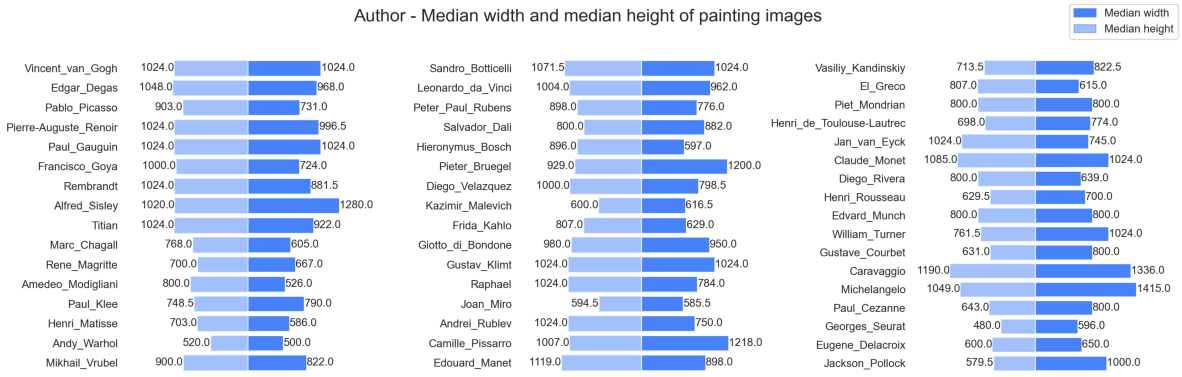

Figure 5: Distribution of median image size across artists

Given the large number of images and authors, as well as the ample size distribution of the images, the dataset provides more than enough material to select style images.

3.3 Pretrained Model

In addition to the input images, this project leverages the VGG-19 model, which is a pre-trained convolutional neural network (CNN) that has been widely used in the field of computer vision for image classification tasks. By using the VGG-19 model as a feature extractor, we can efficiently capture the style and content of input images. This feature extraction process is critical in Neural Style Transfer, as it allows us to separate the content and style information of images, which are then used to generate a new stylized image. Furthermore, using a pre-trained model such as VGG-19 significantly reduces

the computational resources and time required for training our Neural Style Transfer model, allowing us to focus on optimizing the style transfer process itself. The resulting images are compared against images generated by a Fast Style Transfer model.

4 Methodology

We begin by acquiring a general understanding of the the theory behind Neural Style Transfer. Then, we explore different implementations of Neural Style Transfer models in two popular deep learning frameworks: Tensorflow [4] and Pytorch[7]. We fine-tune and examine different parameters for each implementation, and then select the best model based on the resulting stylized images. For this exploration, we use a single-style image, which is presented in Figure 6.

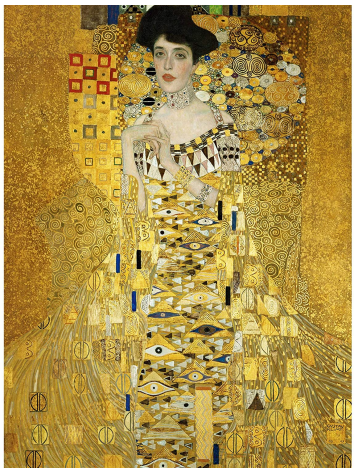

Figure 6: Adele Bloch-Bauer, by Gustav Klimt. Style image used for the exploration phase of Neural Style Transfer

At the end of the exploration phase, we implement the refined model using AWS Sagemaker estimators, leveraging the power of cloud computing to generate high-quality stylized images efficiently, using a selected sequence of style images.

Finally, we compare the generated images using Neural Style Transfer with images produced by using Fast Style Transfer using appropriate metrics.

4.1 Theory of Neural Style Transfer

Neural Style Transfer is a deep learning technique that allows the generation of a new image that combines the content of one input image with the style of another. The core idea behind Neural Style Transfer is to separate the content and style information of an image, and then use this information to generate a new image that preserves the content information while adopting the style of the style image [1].

The content information of an image can be captured by the values of the intermediate feature maps in a deep neural network, such as the VGG-19 model. The style information of an image, on the other hand, can be captured by the means and correlations across the different feature maps in the deep neural network. These means and correlations can be measured using the Gram matrix, which is a matrix that describes the correlations between different feature maps in a given layer of the network [4].

Based on this, given a content image C and a style image S, we want to generate a new image X that combines the content of C with the style of S by minimizing a loss function that consists of two components: a content loss and a style loss. Formally, the loss function for Neural Style Transfer can be expressed as follows:

$$L_{total}(C, S, X) = \alpha L_{content}(C, X) + \beta L_{style}(S, X)$$

The content loss $L_{content}$ measures the difference between the feature representations of C and X, while the style loss L_{style} measures the difference between the Gram matrices of the style representations of S and X. In the formula, α and β are weighting factors that control the relative importance of the content and style losses, respectively [1].

Gradient descent optimization is used to iteratively update the pixel values of X in order to minimize the total loss function. By minimizing the loss function, we can generate a new image that combines the content of C with the style of S, resulting in a high-quality stylized image.

4.2 TensorFlow implementation

4.2.1 Data Preprocessing

The TensorFlow implementation uses a pre-trained model for the feature extraction step, so, no major data preprocessing is necessary. However, the content and style images are resized to a manageable size that can be processed efficiently by the model (512 pixels), while preserving the aspect ratio. In addition, the image values are scaled to the range [0, 1].

4.2.2 Implementation

The implementation uses the intermediate layers of the VGG19 model to get the content and style representations of the image. Based on these style and content tensors, a loss function is defined, which consists of a content loss that measures the difference between the feature representations of the content and generated image, and a style loss that measures the difference between the Gram matrices of the style representations of the style and generated images. To minimize the combined loss function and generate the stylized image, the L-BFGS optimizer is used, which is well-suited for optimizing non-linear functions such as the neural style transfer loss function.

4.2.3 Refinement

Several parameters can be fine-tuned, such as the number of epochs, the steps per epoch, as well as hyperparameters for the loss functions (content weight vs style weight). One downside of this implementation is that it produces a lot of high-frequency artifacts [4], so there is an additional regularization term that can be added to the loss function, called the "total variation loss".

4.3 Pytorch implementation

4.3.1 Data Preprocessing

The Pytorch implementation also uses a pre-trained model for the feature extraction step. The content and style images are also resized to a manageable size that can be processed efficiently by the model (512 pixels), while preserving the aspect ratio. One notable difference with the TensorFlow implementation is that in the Pytorch one, the style and content images need to be of the same size, so the style image is resized to the size of the content image. Finally, the image values are also scaled to the range [0, 1].

4.3.2 Implementation

The PyTorch implementation uses the VGG19 network, similar to the TensorFlow tutorial. However, the PyTorch tutorial uses a different set of layers to extract the content and style representations of the image. Based on these representations, content loss, and a style loss are defined and subsequently optimized via gradient descent using L-BFGS optimizer.

4.3.3 Refinement

Several parameters can also be fine-tuned, in this implementation such as the number of epochs, the number of steps per epoch, as well as hyperparameters for the loss functions (content weight vs style weight).

4.4 Final Neural Style Transfer model

The final Neural Style Transfer model is utilized with various paintings to generate stylized images. To achieve efficiency in the process we use AWS Sagemaker estimators. In particular, the 'Estimator' class of Sagemaker allows running several models simultaneously by spawning multiple training instances. To make this possible from a Jupyter notebook, we ran sequentially the in the "fit" method of the "Estimator" class with the parameter "wait=False". To check the training job completion, we used the Sagemaker Client Session method "describe_training_job".

4.5 Evaluation Metrics

The evaluation of style transfer quality is typically done by comparing the output image to the target style image and measuring the similarity between them. One common metric for evaluating style transfer quality is the mean squared error (MSE) or the root mean squared error (RMSE) between the pixel values of the output image and the target style image. However, these metrics may not be the best choice for evaluating style transfer quality, as they do not necessarily capture the visual similarity between the images.

More recently, there has been a growing interest in using perceptual metrics to evaluate style transfer quality. Perceptual metrics measure the similarity between images based on their perceptual features, such as color, texture, and structure, rather than their pixel values. One popular perceptual metric for evaluating style transfer quality is the Structural Similarity Index (SSIM), which measures the similarity between images based on their luminance, contrast, and structure.

5 Results

5.1 TensorFlow implementation

Figure 7 shows the image generated by the Tensorflow implementation using a style loss weight of 0.01 and a content loss weight of 100000, for different numbers of epochs and different numbers of steps per epoch:

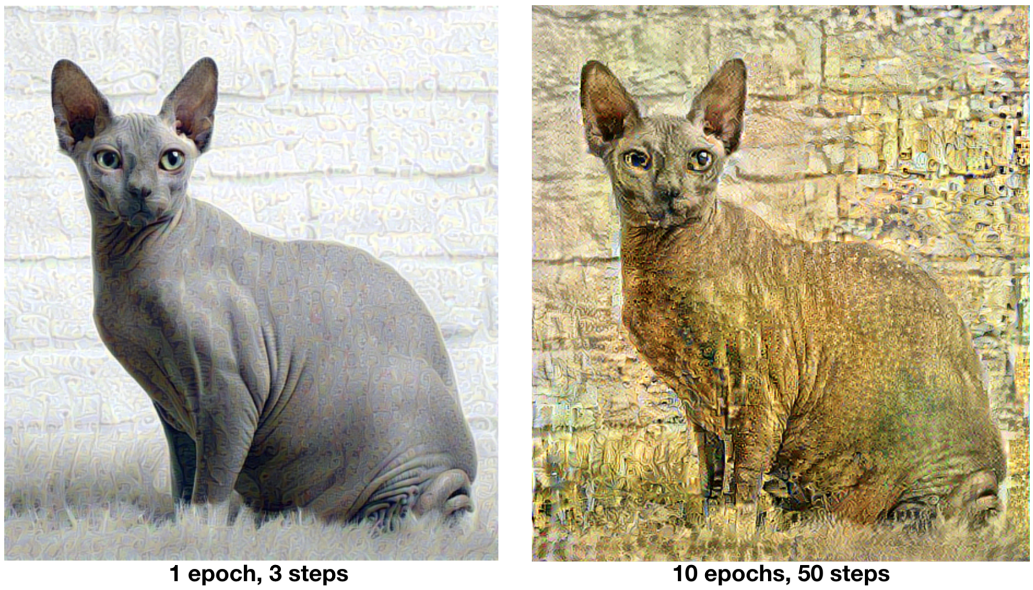

Figure 7: Generated images for different numbers of epochs and different numbers of steps per epoch

As can be seen, the larger the amount of epochs and steps per epoch, the better the style transfer is. Nevertheless, as the number of iterations increases, the TensorFlow implementation produces a lot of high-frequency artifacts. This effect can be seen clearly in Figure 8, where the horizontal and vertical deltas of the original and stylized image are presented:

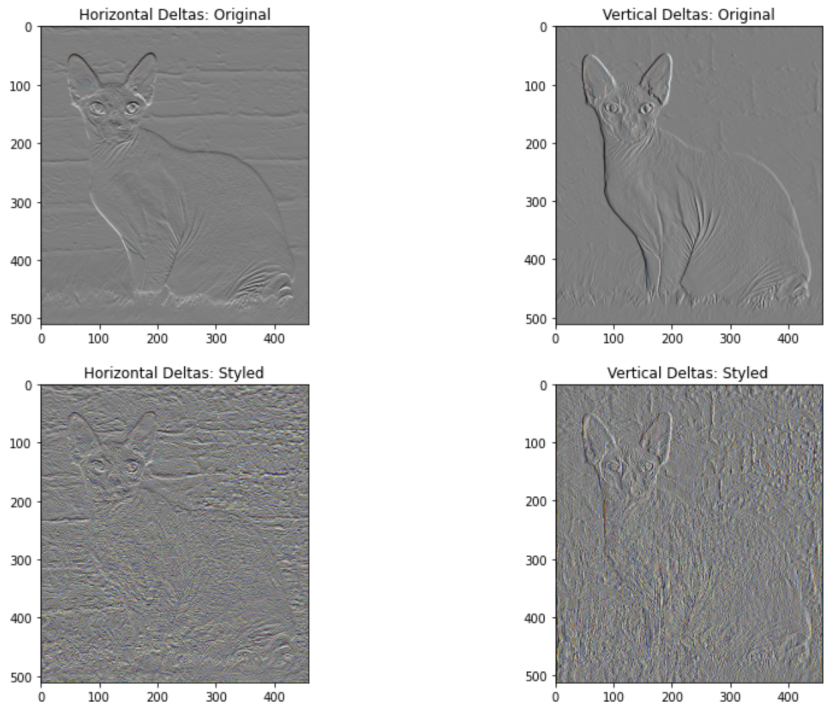

Figure 8: Horizontal and vertical deltas of the original (top) and stylized image (bottom)

To counteract, this effect, we add a total variation component to the loss function. Figure **??** shows the images generated using 10 epochs and 50 steps, after adding a total variation loss as a regularization factor:

Figure 9: Generated images using 10 epochs and 50 steps for different weighting factors of the total variation loss

After applying the regulation based on total variation, the resulting images look smoother compared to the previous ones. However, very high weights for the total variation loss result in blurred sections of the image.

5.2 Pytorch implementation

Figures 10 and 11 show the images obtained by the Pytorch implementation using different epochs, with 300 steps per epoch, a content loss weight of 10, and style loss weights of 1e6 (Figure 10) and 1e7 (Figure 11).

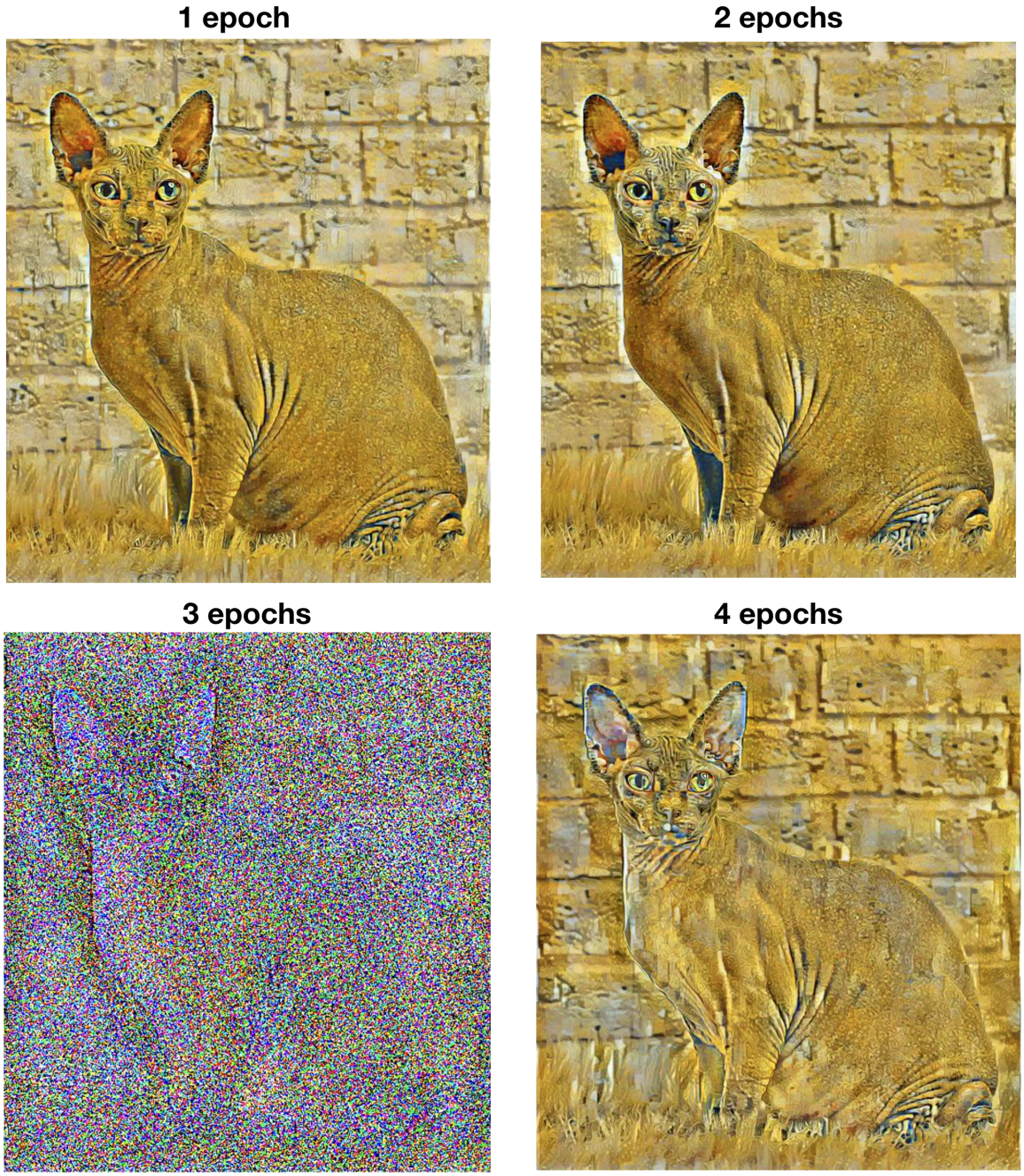

Figure 10: Generated images using different epochs and a style loss weight of 1e6

From Figure 10, we can see that in Epoch 3 there is an optimization issue, and the content and style loss explode. During Epoc 4, it seems that the model manages to reduce the losses, but the quality is not as good as the one produced after 2 epochs.

For Figure 11, we don't observe the losses explode in any of the epochs, and the resulting image is much better after 4 epochs. However, the high weight for the style loss forces us to use a high number of epochs.

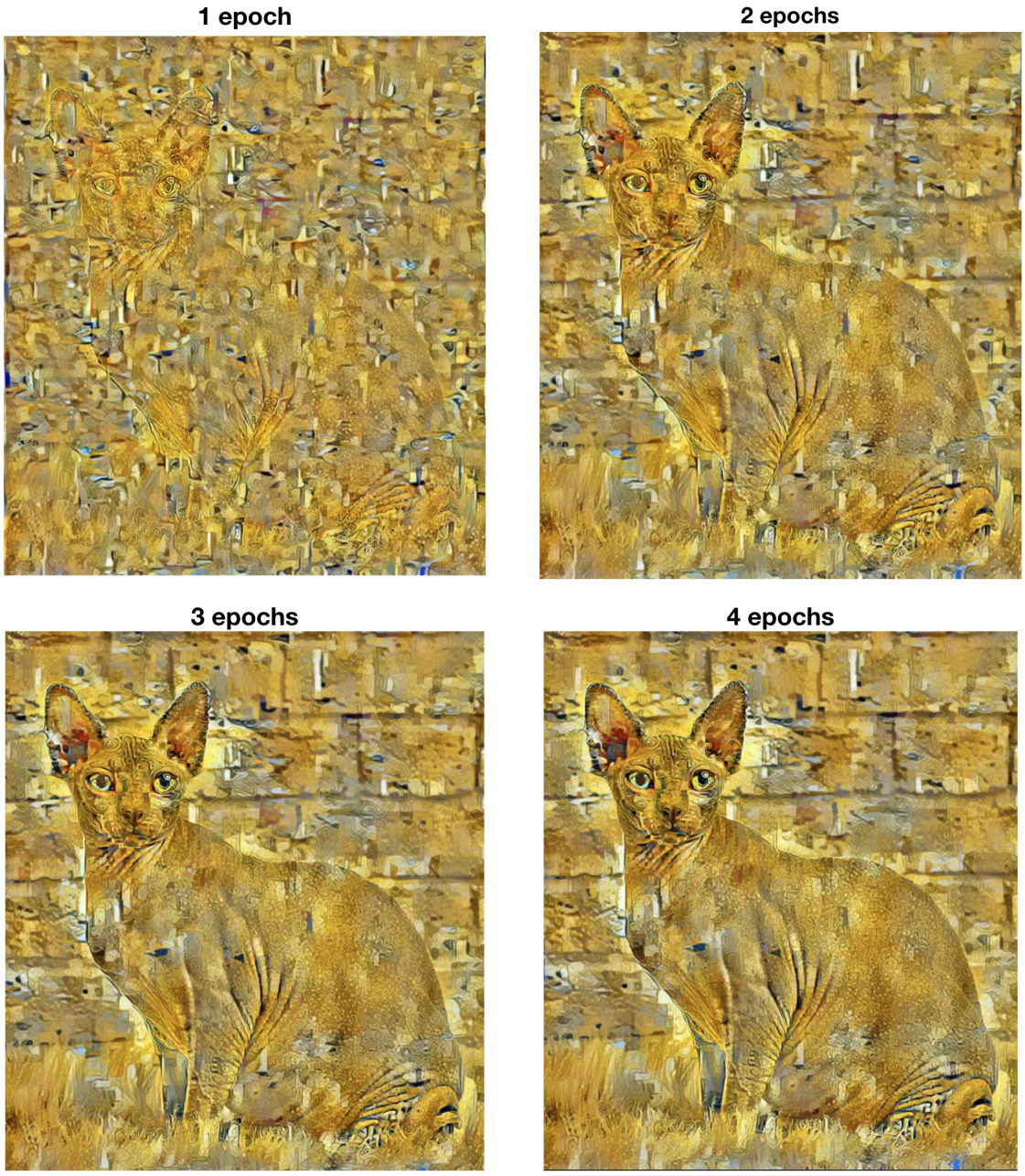

Figure 11: Generated images using different epochs and a style loss weight of 1e7

5.3 Final Neural Style Transfer model

After having explored the different implementations in TensorFlow and Pytorch, fine-tuning multiple parameters, we decided to select the Pytorch implementation with 2 epochs, 300 steps per epoch, content loss weight of 10, and style loss weight of $1e6$. The decision was based on the following factors:

- The images produced by the TensorFlow implementation contain very high-frequency artifacts, and heavily relying on a high weight for the total variation produces blurry sections in the image
- Too many epochs result in higher computational time (each epoch with 300 steps takes at least 20 minutes), and can lead to the explosion of the losses
- A high style loss weight like $1e7$ requires a higher number of epochs, and for some cases, it might not ensure that the content of the original image is preserved

Using ASW Sagemaker estimators, the final Neural Style Transfer model was run using various style images from the paintings dataset. As well, Fast Style Transfer images were generated using a model from TensorFlow Hub [3]. For each image, we calculated the Mean Squared Error (MSE) and the Structural Similarity Index (SSIM), and computed the percentage improvement for each image as follows:

$$MSE_improvement = \frac{(MSE_Fast\_Style - MSE\_Neural\_Style)}{MSE\_Fast\_Style} \times 100$$

$$SSIM_improvement = \frac{(SSIM_Neural\_Style - SSIM\_Fast\_Style)}{SSIM\_Fast\_Style} \times 100$$

Figure 12 shows the improvements in the SSIM and MSE for the images generated using Neural Style Transfer, vs the images generated with Fast Style Transfer.

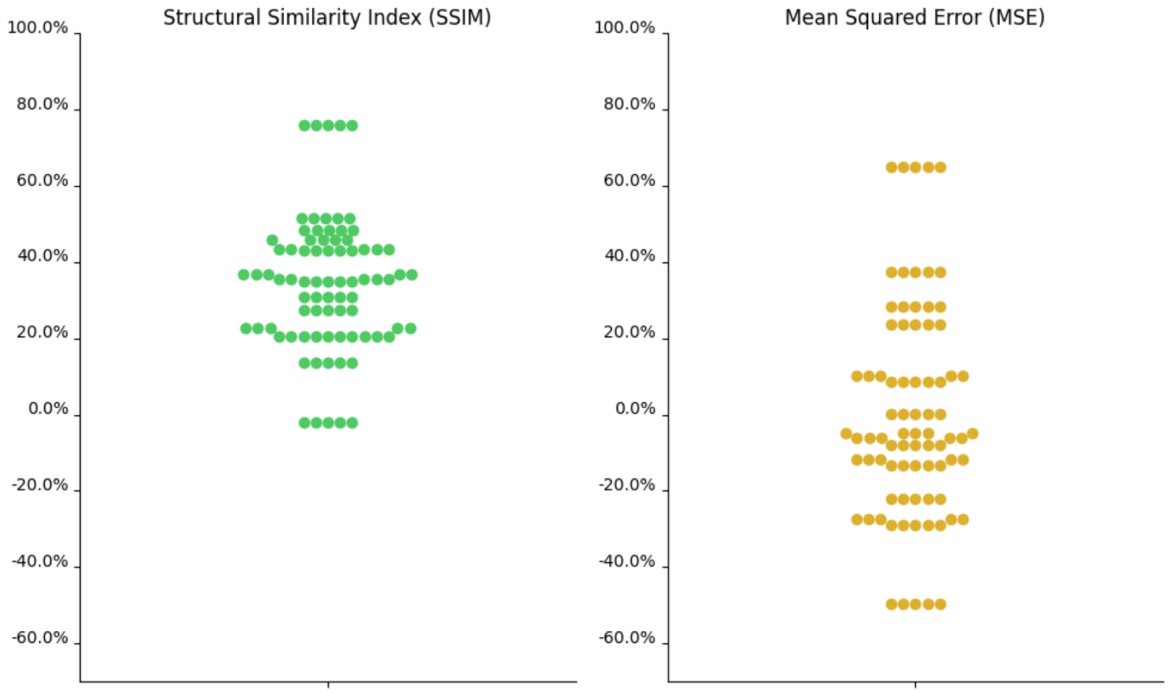

Figure 12: Percentage improvement in the SSIM and MSE metrics when using Neural Style Transfer to generate stylized images

From Figure 12 it is noticeable that there is a large improvement in the Structural Similarity Index (SSIM). We cannot say the same about the Mean Squared Error (MSE), but as mentioned before, the MSE may not be the best choice for evaluating style transfer quality, as it does not necessarily capture the visual similarity.

Finally, figures 13 to 15 show the resulting images obtained and provide a comparison of the Neural Style Transfer vs the Fast Style Transfer. By visually inspecting the images it can be observed that the Neural Style Transfer better preserves the original content of the image.

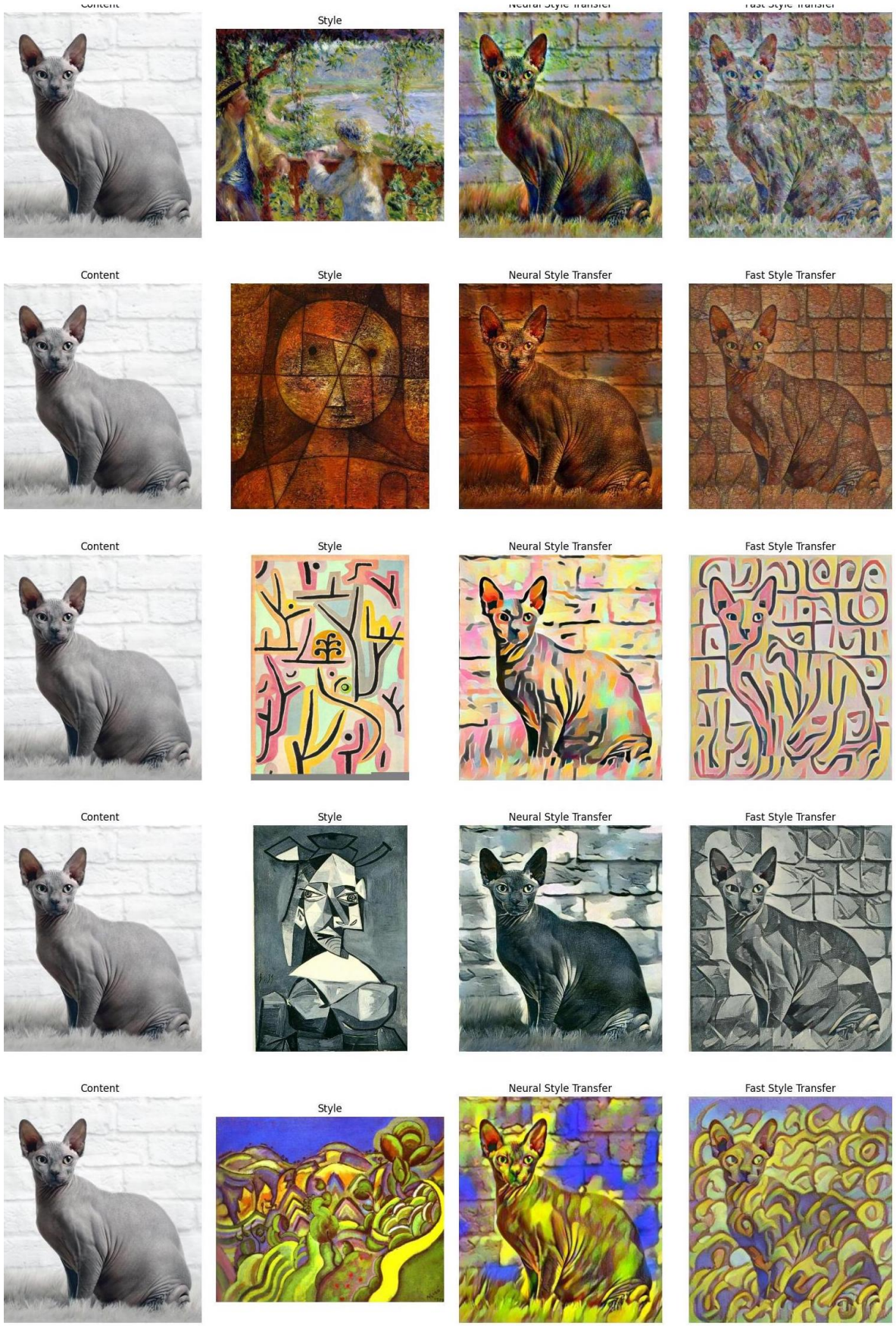

Figure 13: Results after applying the refined model on various style images - Part 1/3

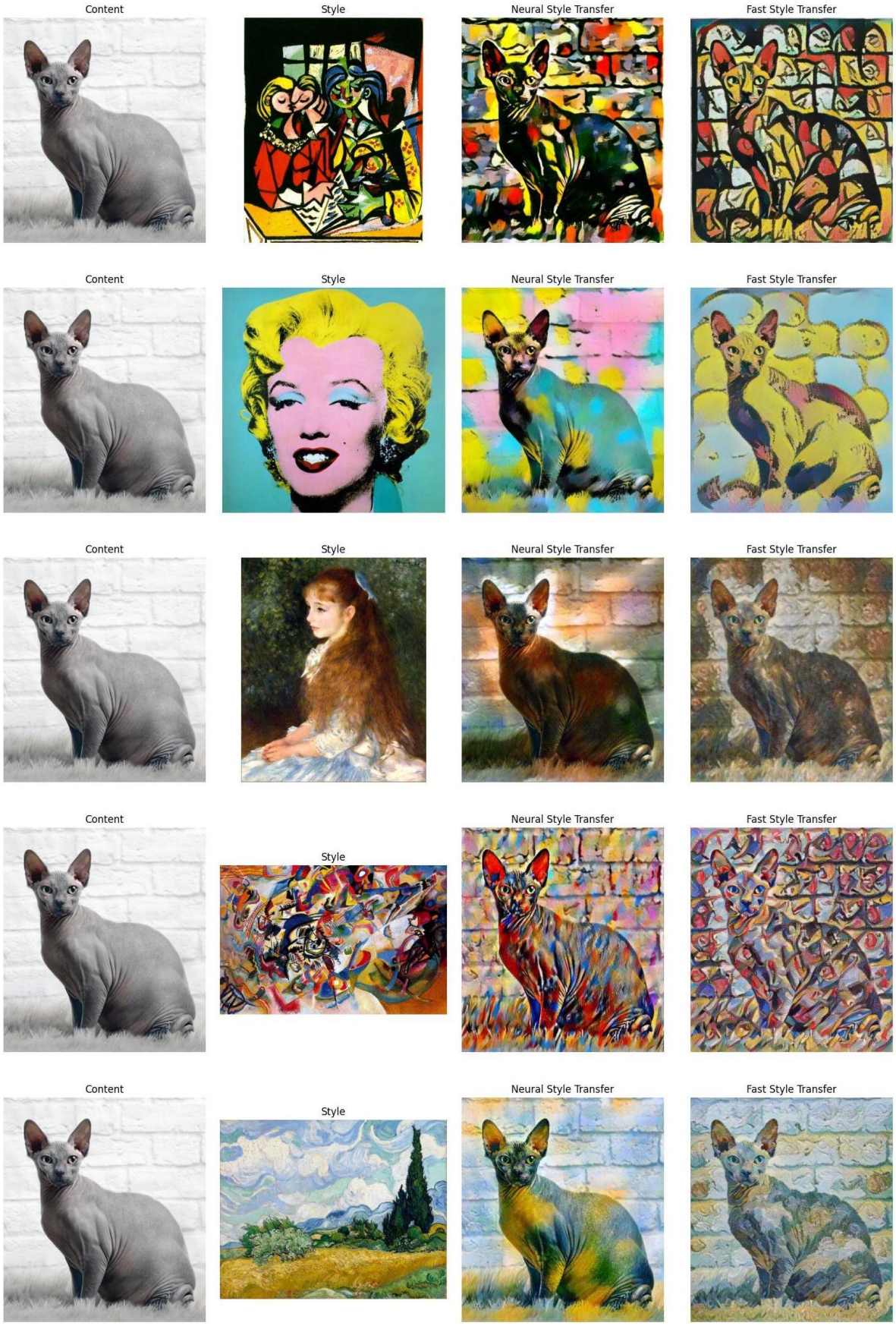

Figure 14: Results after applying the refined model on various style images - Part 2/3

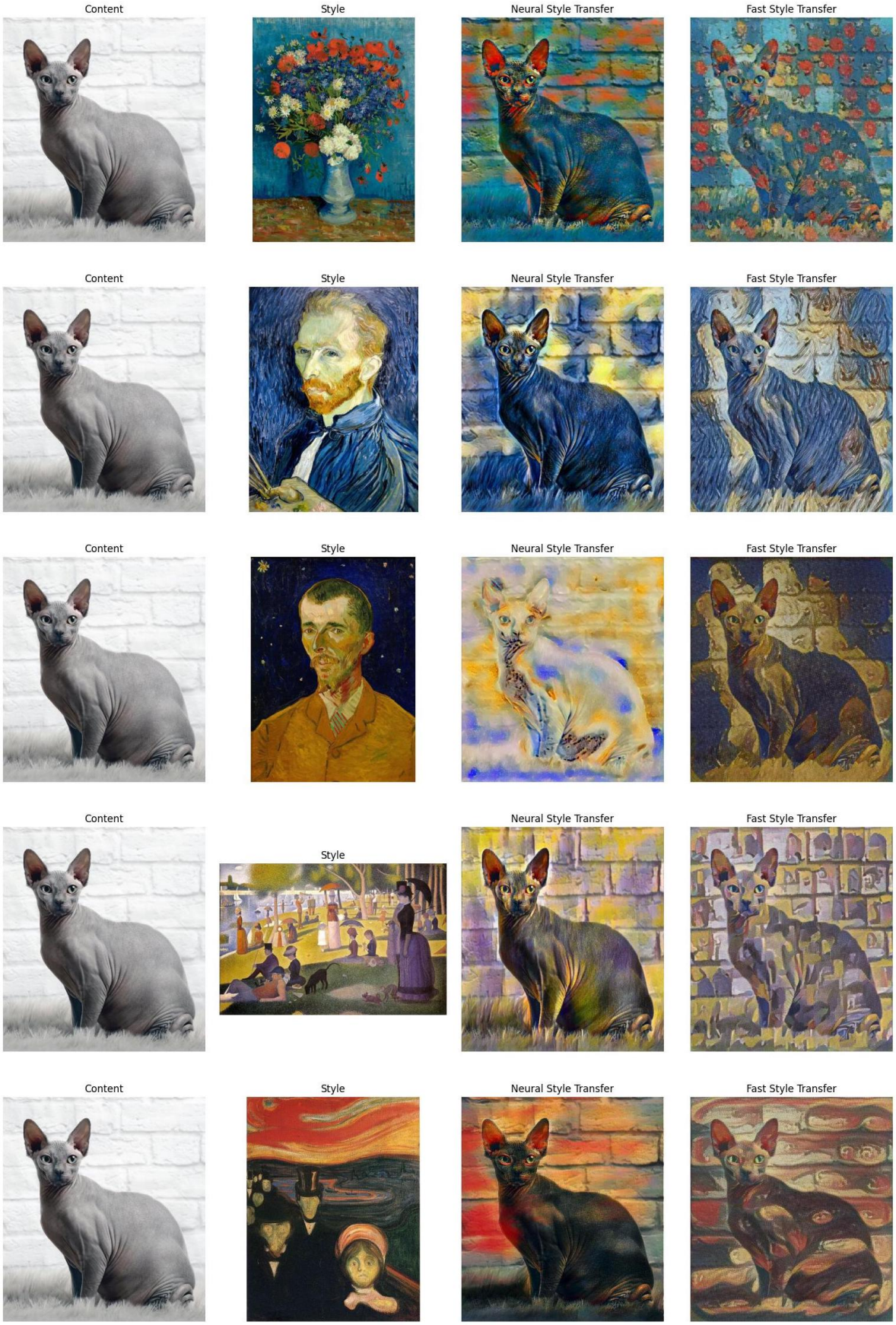

Figure 15: Results after applying the refined model on various style images - Part 3/3

6 Conclusion

In conclusion, this project successfully implemented a traditional Neural Style Transfer model in AWS Sagemaker that produces higher quality stylized images than Fast Style Transfer models.

The methodology involved exploring different implementations of Neural Style Transfer models in Tensorflow and Pytorch, fine-tuning and examining different parameters for each implementation, and selecting the best model based on the resulting stylized images. The resulting model was then implemented using AWS Sagemaker estimators, leveraging the power of cloud computing to generate high-quality stylized images efficiently. Evaluation metrics such as MSE and SSIM were used to compare the generated images using Neural Style Transfer against images produced using Fast Style Transfer.

From the results, it can be concluded that the Neural Style Transfer better preserves the original content of the image and has a large improvement in the Structural Similarity Index. This project showcases the potential of traditional Neural Style Transfer models for generating high-quality stylized images and provides insights into the strengths and limitations of the commonly used Fast Style Transfer technique.

In future work, the following possibilities could be explored:

- Test the Neural Transfer Models using other content images
- Use a variation of the content loss and style loss for different epochs
- Add a total variation loss to the Pytorch implementation
- Save the images with a higher frequency, and do early stopping when the losses explode
- Perform a more extensive hyperparameter tuning of the different parameters of the Neural Transfer Models

References

[1] 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.

[2] *How to style transfer your own images*. Feb. 2020. URL: https://godatadriven.com/blog/how-to-style-transfer-your-own-images/.

[3] *Fast style transfer for arbitrary styles nbsp;: nbsp; tensorflow hub*. URL: https://www.tensorflow.org/hub/tutorials/tf2_arbitrary_image_stylization.

[4] *Neural style transfer nbsp;: nbsp; Tensorflow Core*. URL: https://www.tensorflow.org/tutorials/generative/style_transfer.

[5] *Caring for Your Sphynx Cat*. URL: https://www.catwatchnewsletter.com/features/caring-for-your-sphynx-cat/.

[6] *Best Artworks of All Time*. URL: https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time.

[7] Alexis Jacq. *NEURAL TRANSFER USING PYTORCH*. URL: https://pytorch.org/tutorials/advanced/neural_style_tutorial.html.