Generating Monet Paintings with CycleGAN

Group 20
Aashutosh Ganesh (s1055287)
Thomas Rood (s1005156)
Pascal Schröder (s1062138)
Radboud University
Nijmegen, The Netherlands

1 INTRODUCTION

Since its introduction, the neural style transfer algorithm by Gatys et al. [4] has brought about a new computational understanding of the style of an image. It utilised the Gramian matrix in a dual optimisation equation to generate an image. Further work has built upon the original equation to add, for instance, regularisation [14], segmentation guided maps [21], and whitening and colouring transformations [13], all in efforts to improve image quality and reduce computation time. Recently, approaches using generative adversarial networks (GANs), in which a generator and discriminator network pit against each other, were amongst the most promising for style transfer [5]. Popular architectures include Siamese GANs [8] and CycleGAN [22].

The goal of the Kaggle challenge "I'm Something of a Painter Myself" [9] lies in using GANs to transfer the style of Claude Monet's artworks to a set of photographs of natural scenes (referred to as 'photos' or 'natural images'). To this extent, a set of 300 Monet paintings and 7038 photos are provided as input data. The quality of style transfer is evaluated by means of the Memorisation-informed Fréchet Inception Distance (MiFID) score, which is a variation of the Fréchet Inception Distance [7] that penalises memorisation. We explore and evaluate how several modifications to a baseline model impact the MiFID test score.

2 METHODS

In this section we introduce the CycleGAN model, which served as the foundation for our experiments. Because our strategy involved short and iterative development cycles, it must be noted that the adaptions we undertook were not independent of each other. Rather, certain ones were chosen as a result of others failing to yield improved results.

2.1 Baseline Model

We started out with a publicly available CycleGAN model created for this challenge, which had achieved a reasonable final MiFID score of 45.82 [2]. The kernel for this model already utilised data augmentation in the form of random horizontal flips on both Monet and natural image input data, and was optimised to take advantage of the Tensor Processing Units (TPUs) provided as accelerators for this challenge. Other than that, the kernel followed the original CycleGAN implementation, allowing us to add more optimisations.

2.2 ResNet vs. U-Net Generators

Our initial experiment revolved around choosing the architecture of the generator networks, to see which worked best. Two different approaches found in the literature are either based on U-Net [16] or ResNet [6]. U-Net was already used in the baseline model, thus we implemented a generator using a ResNet architecture based on the official Keras documentation [15], and evaluated the resulting model against the baseline.

2.3 Data Augmentation

We investigated the use of additional data augmentation. Since the base model successfully incorporated horizontal flip augmentations, we saw it as a promising direction to apply more severe augmentations to the data.

First, we aided the present augmentations with random vertical flipping. An often posed argument against the usage of vertical flipping is that it produces unrealistic images, for example by turning a landscape picture upside down. However, we hypothesised that the style of an image is independent of the orientation of the semantic content in the image, and that the style transfer task would thus be unaffected by this problem.

Further, we attempted hue, saturation and brightness alterations as possible augmentations, which we hypothesised to be able to contribute to the robustness of the model by aiding the model to become invariant of global picture statistics.

2.4 Additional Data

We noted a great imbalance in the provided data between the Monet paintings (300 images) and the photos (7038 images). To counteract this imbalance, we provided our model with more Monet paintings from another dataset found on Kaggle [12]. Conveniently, this dataset had the same image size and format as the original dataset provided in the challenge, meaning that no additional processing was required. Combining this with the original 300 Monet paintings yielded a dataset of 1667 Monet images for our model to train on. We tested two runs with this dataset. For the first one we concatenated the 1667 non-augmented images with augmented versions with 50% probability of horizontal flips, and 50% probability of vertical flips, yielding 3334 Monet images in total. For the second run, we only used 1667 images, but applied random data augmentation with the same probabilities to it instead of concatenating. This yielded a dataset with 75% augmented images on average.

2.5 Monitoring Training Through MiFID

Evaluating the convergence of the model during training was a complicated problem, since the CycleGAN model consisted of four individual networks, each with individual loss functions that ade up the total loss. Since in a GAN, the generator loss depends on the

quality of the discriminator and vice versa, the absolute quality of the model cannot be trivially derived from the losses. The relationship between the four different losses of the CycleGAN model and the model performance was therefore not known.

With this in mind, we determined that it would be hard to track the convergence of the model while training using loss functions only. Instead, we proposed using the MiFID score. From its usage as the scoring metric for the challenge submissions we hypothesised that it could adequately measure the quality of the model during training time as well. To this end, we adapted the code from [11] and [20]. Because of limited memory on the Kaggle machines, we used the training dataset to compute the MiFID at the end of each epoch.

In addition, we added the possibility to save the model with the best MiFID score, which could prove useful in the case of fluctuations in model quality between epochs. For more details on monitoring training through MiFID, see Appendix A.2.

2.6 Learning Rate

During our experiments, we observed that the per-epoch MiFID training score was far from converging after training for 100 epochs. Given this observation, we hypothesised that a better score could be achieved when the model could run for longer than 100 epochs. Unfortunately, for most of our experiments, 100 epochs was the maximum number of epochs for which the execution time was still in the allowed time limit of 3 hours, as imposed by the challenge. To circumvent this, we hypothesised that faster convergence may be achieved by increasing the learning rate. As a first try, we increased the learning rate from the default 2×10^{-4} to 3×10^{-4}.

For more granular control, we reasoned that an initially high learning rate with subsequent decay may be even better suited. To this end, we implemented a learning rate scheduler that decays the learning rate in a linear fashion. More specifically, the learning rate would stay constant for a specified number of epochs at a value of 4×10^{-4}, and linearly decay until it reached 0 at the maximum amount of epochs. We first trained a model with 25 epochs of constant rate, and 75 additional epochs of decaying rate. In addition, we ran an experiment with 50 epochs of constant rate and 100 epochs of decaying rate, for a total of 150 epochs.

2.7 Batch Size

An interesting observation we made was that all notebooks for the Kaggle challenge seemed to use a batch size of 1. Generally speaking, using mini-batches of a larger size is associated with more stable training (due to averaging the gradients over multiple datapoints), as well as being more efficient to compute [17]. In fact, for the TPU accelerators provided for this challenge, a batch size of 1024 is recommended for maximal efficiency [10].

We noted that because of the unpaired nature of the data, the batch size could only be a common divisor of the sizes of both the photo and painting datasets. Otherwise, the last batch would need to be discarded or padded, in order to prevent shape mismatches. We therefore tried the greatest common divisor as a batch size, which resulted in a batch size of 6.

2.8 Experiments

With the aforementioned general descriptions of optimisations, we conducted numerous experiments. Given that the challenge posed a 3 hour run time limit on TPU accelerated notebooks, the number of epochs we could train each model for was often limited to 100 epochs using batch size 1. In certain cases, such as the addition of more data, we had to lower the number of training epochs to stay within the time limit.

Later in our experimentation, we noted that we could increase the total number of epochs for the batch size of 6 without exceeding the time limit, since execution time per epoch was greatly decreased. Because we only discovered the potential of this late in our experimentation phase, most of our runs featured a batch size of 1.

In addition, we conducted a control experiment, in which we evaluated the MiFID score on the non-stylised, default photo images.

3 RESULTS

In this section we will describe the results of the discussed optimisations. In Table 1, the run time and MiFID scores of the models are summarised. In Figure 1, three images stylised by the best performing model are shown accompanied by the original images and examples of Monet paintings.

Model	# Epochs	Run time (s)	MiFID
Control	N/A	N/A	78.96
Baseline [2]	50	1h49m	45.82
ResNET	15	N/A[1]	67.60[2]
Baseline + BS6	100	52m	49.38
Baseline + BS6 + HSB	100	51m	59.51
Baseline + BS6 + VF	100	1h00m	45.94
Baseline + VF	100	2h53m	41.71
Baseline + VF + LR3E-4	100	2h59m	52.04
Baseline + VF + LRD	100	2h57m	43.97
Baseline + BS6 + VF	**400**	**2h13m**	**39.94**
Baseline + VF + DAT	20	3h12m[3]	46.30
Baseline + VF + DAT	40	3h09m[3]	47.77
Baseline + VF + MS	100	3h31m[3]	44.35
Baseline + VF + LRD	150	4h12m[3]	41.30

Table 1: Experiments with corresponding run time and MiFID score. Abbreviations: BS6 = batch size of 6, VF = vertical flip augmentations, HSB = hue, saturation and brightness augmentations, DAT = additional Monet data, MS = MiFID-based model saving, LR3E-4 = learning rate 3E-4, LRD = learning rate decay. [1]Time measured by Kaggle was 4.6 seconds, which we assumed to be wrong. [2]Score was obtained at 15 epochs of run time, since running for longer was infeasible due to the model not working with TPU acceleration. [3]Time limit of 3 hours exceeded.

(a) Original Images

(b) Generated Monet Paintings (Best Model)

(c) Exemplary True Monet Paintings

Figure 1: Comparison of original images, output images of the best performing model (Baseline+BS6+VF with 400 epochs), and example Monet paintings.

ResNet vs. U-Net Generators. The ResNet generator implementation proved difficult due to the necessary reflection padding not working with TPU acceleration. Therefore, we were unable to train the model for a meaningful amount of time. The 15 epochs that were trained did not show a promising result, with a MiFID score of 67.60. Thus, we continued with the U-Net architecture.

Batch Size. We observed that using a batch size of 6 performed considerably worse than using a batch size of 1. When using vertical flipping augmentations and training for 100 epochs, a batch size of 6 yielded a score of 45.94, while a batch size of 1 achieved 41.71. The same model of batch size 6 trained for 400 epochs showed an improved score with respect to both those models, yielding our best model with a score of 39.94.

Data Augmentation. A large decrease in score was found for the usage of hue, saturation and brightness augmentations with respect to the Baseline + BS6 model, with the augmentations resulting in a MiFID score of 59.51 (as opposed to 49.38). However, only adding random vertical flipping did improve the score with respect to the Baseline + BS6 model. Combined with the a batch size of 1, this yielded our second best model with a score of 41.71.

Learning Rate. For the experiment where we used a learning rate of 3×10^{-4}, we observed a large decrease in score with respect to the Baseline + VF model, with a score of 52.04.

With linear learning rate decay, a model trained for 100 epochs achieved a score of 43.97. As this model was based on the best run at that time with a score of 41.7, this was a decline in model quality.

On the other hand, the learning rate scheduled run with 150 epochs of training achieved a score of 41.3. However, we noted that this run exceeded the maximum execution time by over an hour. Although accepted by the leaderboard, we treated this model as violating the rules of the challenge, and thus disqualified it from being a valid entry.

Additional Data. The increased number of data points meant that each epoch would take longer to compute. For the first run with 3334 paintings in total, we could not train for longer than 20 epochs. For the second run with 1667 paintings we reached the limit with 40 epochs. We noted that both runs still exceeded the maximum run time, albeit by a much smaller margin than the 150 epoch model with learning rate decay. Achieving a MiFID score of 46.3 and 47.8 respectively, these models performed considerably worse than the model they were based on.

Monitoring Training Through MiFID. In Figure 2 the MiFID score per epoch progression of the Baseline + VF + MS model is displayed. We observed a steadily decreasing trend in the score, even though individual epoch scores could deviate quite a lot. Notably, we observed no signs of convergence near the end of the 100 epochs, meaning that longer training would likely result in a better score. Using MiFID based model saving, the model at epoch 94 was saved, which achieved a score of 44.35, performing worse than the model without MiFID model saving.

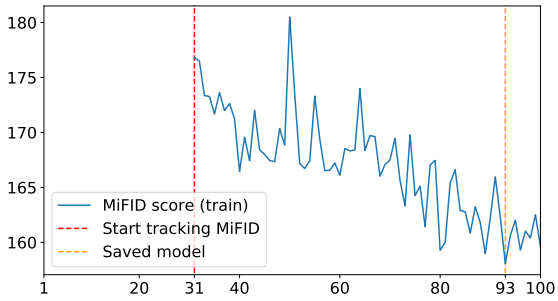

Figure 2: Training MiFID score of the Baseline + VF + MS model. Note that the scale of MiFID scores differs from that of the official Kaggle scores (see Appendix A.2 for details).

Subjective Visual Quality. The generated images indeed incorporated some features akin to Monet's style in a subjective sense, with washed out yet contrasting colours as seen in Figure 1. We do however notice some artefacts, especially visible in the left and right images. These make it evident to a human viewer that the generated images are not Monet paintings. See Appendix A.3 for more examples of different models.

4 DISCUSSION

Our experiments resulted in a wide range of interesting results. First of all, we noted how the hue, brightness and saturation augmentations resulted in a large decrease in score. We note that this is likely due to the fact that these types of augmentations, as opposed to the flip augmentations, alter the statistics of the images. Such augmentations might prove useful in a classification tasks to increase the amount of input data. However, in style transfer this may impact the unique artist style the model is supposed to extract. The addition of the random vertical flip augmentation on the other hand showed an increase in score with respect to the models without it, and consequently was part of our best model.

Combining the good results from the 150 epoch learning rate decay run, as well as the MiFID score monitored during training, we established that the upper bound of 100 epochs most of our models were subject to did not suffice for them to reach full convergence. Subsequently, we hypothesised that with longer training a better score could likely be achieved. We expected to possibly achieve this by using the learning rate decay within 100 epochs, allowing more efficient learning in the same amount of time. Contrary to our expectations, the results showed a decrease in score. However, this could be due to the small selection of specific initial learning rates and decay rates we chose to try with the limited amount of resources at hand.

To our surprise, a batch size of 6 did improve the quality of the model when executed for a large amount of epochs. This was somewhat unexpected, since the majority of implementations we found on Kaggle, the original CycleGAN paper [22], and the official Keras implementation [15] used a batch size of 1.

We found that the model was not converging within 100 epochs with a batch size of 1. Because the more efficient computation provided by the batch size of 6 allowed more training without exceeding the time limit, it made sense that it increased the score.

Limitations. We note the following limitations to our approach. First off, the experiments that we ran resemble a greedy search through the space of possible optimisations, rather than a grid search approach. Being more structured, the latter may have revealed configurations that could outperform the current best model. However, given the plethora of potential improvements and long training times, we argue that the conducted approach was a good balance between optimality and time.

Related to this, it would have been interesting to see the effect of the optimisations when taking the Baseline + BS6 + VF model trained for 400 epochs as a basis. The optimisations that did not prove useful for the models trained for 100 epochs might have benefited this model, and improved the score even further. However, given the limited time and late discovery of this improvement, we were not able to pursue this.

Secondly, the usage of additional data, although allowed, gave rise to the risk of training on Monet paintings that were also in the test set used by Kaggle to compute the score. Since there is a limited amount of paintings Monet made, this was a realistic concern. However, the model did not outperform the Baseline + VF model, meaning that this concern is not of importance for the final model.

Finally, we noted that although a reasonably high score was obtained, the visual quality of the stylised images was still lacking, in that noisy patterns appeared which negatively affected the Monet-likeness of the images.

5 CONCLUSION

In conclusion, we proposed a large set of optimisations to an existing CycleGAN model. The results showed that augmenting the data with random horizontal and vertical flips, combined with an increased batch size of 6, constituted our best model when trained for 400 epochs. This model achieved a MiFID score of 39.94. Based on the results from the run with 150 epoch using learning rate decay and the MiFID score monitored during training, we found that the main limiting factor for our models seemed to be the time constraint imposed by Kaggle. The success of our best model is therefore hypothesised to be caused by the higher batch size allowing more efficient training in the prescribed time limit. Finally, we found that, although we achieved a comparatively high score, the subjective visual quality of the images was degraded due to artificial-looking repeating patterns in the stylised images.

For future work, we advise the exploration of even higher batch sizes in combination with the optimisations proposed in this report.

6 AUTHOR CONTRIBUTIONS

Ganesh	Rood	Schröder
Data Analysis	Batch Size	ResNet
Data Augmentation	MiFID	Learning Rate Decay
Baseline	Baseline	Additional Data
Report	Report	Report

7 EVALUATION OF THE PROCESS

We look back on the project as an insightful experience in which we learned a lot. The collaboration went smoothly, as all authors were excited and interested in the task at hand. In line with this, we met weekly to evaluate results, propose new avenues to explore and divide the work accordingly. We did find it hard at times to keep a clear overview of the code base, versions and results, because we found ourselves bound to working with numerous Kaggle notebooks.

8 EVALUATION OF THE SUPERVISION

We were content with the regular supervisor meetings, as they ensured keeping the course structured and motivated progress. Especially the coach meetings were very useful. Our coach Chris Kamphuis gave valuable insights to discussion points, and meeting with the other groups enabled cross-group insights which enriched the project. However, we did feel that individually meeting with the teacher was less insightful, mainly due to the time limit of 10 minutes which prevented a fruitful discussion.

REFERENCES

[1] Animesh. 2020. *Monet GAN.* https://www.kaggle.com/animesh2099/monet-gan (Accessed on 2021-03-13).

[2] Joaquín Bengochea. 2021. *CycleGAN_v1.* https://www.kaggle.com/joackobengochea/cyclegan-with-data-augmentation?scriptVersionId=52447338 (Accessed on 2021-03-11).

[3] Eric. 2020. *GAN Submission.* https://www.kaggle.com/enm185/gan-submission (Accessed on 2021-03-13).

[4] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A Neural Algorithm of Artistic Style. arXiv:1508.06576 [cs.CV]

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'14)*. MIT Press, Cambridge, MA, USA, 2672–2680.

[6] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[7] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6629–6640.

[8] Chih-Chung Hsu, Chia-Wen Lin, Weng-Tai Su, and Gene Cheung. 2019. SiGAN: Siamese Generative Adversarial Network for Identity-Preserving Face Hallucination. *IEEE Transactions on Image Processing* 28, 12 (Dec 2019), 6225–6236. https://doi.org/10.1109/tip.2019.2924554

[9] Kaggle. 2021. *I'm Something of a Painter Myself.* https://www.kaggle.com/c/gan-getting-started (Accessed on 2021-03-11).

[10] Kaggle. 2021. *Tensor Processing Units (TPUs).* https://www.kaggle.com/docs/tpu (Accessed on 2021-03-11).

[11] Wendy Kan. 2020. *Demo MiFID metric for Dog image generation comp.* https://www.kaggle.com/wendykan/demo-mifid-metric-for-dog-image-generation-comp (Accessed on 2021-03-13).

[12] Mahn Lab. 2020. *monet_tfrecords_extdata.* https://www.kaggle.com/doanquanvietnamca/monet-tfrecords-extdata (Accessed on 2021-03-11).

[13] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. 2017. Universal Style Transfer via Feature Transforms. arXiv:1705.08086 [cs.CV]

[14] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. 2017. Demystifying Neural Style Transfer. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2230–2236. https://doi.org/10.24963/ijcai.2017/310

[15] Aakash Kumar Nain. 2020. *CycleGAN.* https://keras.io/examples/generative/cyclegan/#build-the-generator (Accessed on 2021-03-11).

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (Eds.). Springer International Publishing, Cham, 234–241.

[17] Kevin Shen. 2018. *Effect of batch size on training dynamics.* https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e (Accessed on 2021-03-13).

[18] Tensorflow. 2021. *tf.image.random_hue.* https://www.tensorflow.org/api_docs/python/tf/image/random_hue (Accessed on 2021-03-14).

[19] Tensorflow. 2021. *tf.image.random_saturation.* https://www.tensorflow.org/api_docs/python/tf/image/random_saturation (Accessed on 2021-03-14).

[20] UnfriendlyAI. 2021. *CycleGAN with DG pretraining.* https://www.kaggle.com/unfriendlyai/cyclegan-with-dg-pretraining (Accessed on 2021-03-13).

[21] Huihuang Zhao, Paul Rosin, and Yu-Kun Lai. 2020. Automatic Semantic Style Transfer using Deep Convolutional Neural Networks and Soft Masks. *The Visual Computer* 36 (07 2020). https://doi.org/10.1007/s00371-019-01726-2

[22] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2020. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. arXiv:1703.10593 [cs.CV]

A APPENDIX

A.1 Data Augmentation

For reproducibility, we include the details on how the hue, saturation and brightness augmentations were performed:

Hue. The hue of the whole image was adjusted by a random amount. To this extend, the `tf.image.random_hue` function was used. This function converts RGB image to a HSV space, and changes the hue value by a certain number. The parameter determining how much the resulting hue value could maximally differ from the original value, `max_delta`, was set to 0.5, being the maximum allowed value (see [18]).

Saturation. Similar to the hue channel, the saturation channel also allowed an additional means to augment data. A careful balance needed to be struck in order to use this augmentation as it could alter the feature statistics significantly. In the `tf.image.random_saturation` function, a saturation factor is utilised to randomly scale the saturation channel. While the documentation suggests the usage of a range from 5 to 10 [19], we used a range from 0.8 to 1.2 as we believed that a larger range for altering the saturation might cause artefacts.

Brightness. The pixel value of each pixel in the image was adjusted with a random amount; either increasing or decreasing the brightness. To this extent, the `tf.image.random_brightness` function was used. The parameter determining how much the resulting hue value could maximally differ from the original value, `max_delta`, was set to 0.5.

A.2 Monitoring Training Through MiFID

As stated in the report, we chose to implement the MiFID score in the training procedure as it provided a single interpretable score to evaluate the quality of the model. Another possibility would have been to compute the average over all of the loss values. However, the problem with this was that, because the loss functions of the discriminators and generators consisted of different components, their scales were different. This would mean that when taking the average, the quality of generators would weigh more strongly than that of the discriminators, as their loss values were typically in a higher range. Instead, we agreed that the MiFID would be a more reliable solution.

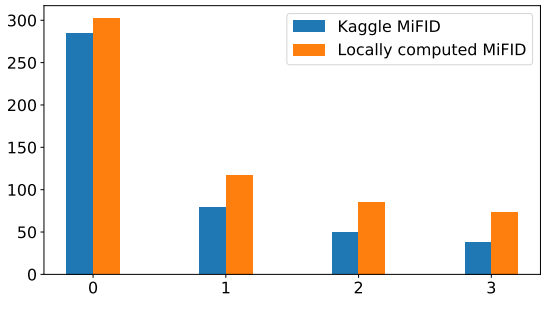

Figure 3: Comparison of MiFID scores as computed by Kaggle and by our implementation adapted from [11]. Datapoints 0-3 respectively represent the model from [3], the Control model, the Baseline + BS6 model and the model from [1].

In order to manually compute the MiFID, we used the code from [11]. After performing the necessary alterations to the code to comply with the data of our challenge, we tested whether the

MiFID scores produced by this algorithm matched those computed by Kaggle. To this end, we ran the output of a selection of notebooks with a broad range of different MiFID scores. We were limited by the fact that we did not have access to the test data that Kaggle would be using to compute the MiFID scores, but still expected relatively similar scores. From the results of the comparison in Figure 3, we can see that although there seemed to be a difference between the scores that is inversely proportional to the score magnitude, the within-score trend seemed to be very similar. This was good news, because it meant that the scores computed by the code could be interpreted in the same way as the Kaggle score, only with a different range of values.

To adapt the MiFID code such that we could run it as efficiently as possible at each epoch, we used the code from [20].

Limitations. In the report we mentioned that we used the training set to compute the MiFID at the end of each epoch. We note that this is a limitation as by not using validation data for this purpose, detecting overfitting would likely be impossible. However, because of the limited amount of memory on the Kaggle machines, we opted to reuse the training set and only see the measure as a tool to check whether the model was still improving on the training data.

In addition, we found the computation of the MiFID score to be computationally expensive. Running the MiFID at the end of an epoch would take roughly 30 seconds, which for 100 epochs would mean an additional 50 minutes of compute time. Since the allowed training time was limited, we only started computing the MiFID score for every epoch after the first 30 epochs had passed. This was done based on the assumption that no problems with training would be encountered this early in the training phase.

A.3 Additional Painting Generation Examples

In figure 4 we included the painting generation results for our top-3 best models. It is interesting to see how the lower performing models show a large amount of checkerboard artefacts. Comparing between the second-to-best and the best model (figures 4c and 4b, respectively), we can clearly see how the checkerboard artefacts for the left and middle image seem to be resolved. We do note that some other artefacts in the left image remain, but overall it is a large improvement. It is additionally interesting to see how for the Baseline + VF + LRD model (figure 4d), the left and right images are clearly manipulated, but the middle image seems largely unaffected by the transformation. We are unsure of the cause of this phenomenon, but it could be interesting for future work to look into this issue, as the underlying cause might be preventing the model from optimally converting all images, thus degrading model quality.

A.4 Code

All reported model kernels and other supplementary code is publicly available at https://github.com/verrannt/i-am-a-painter.

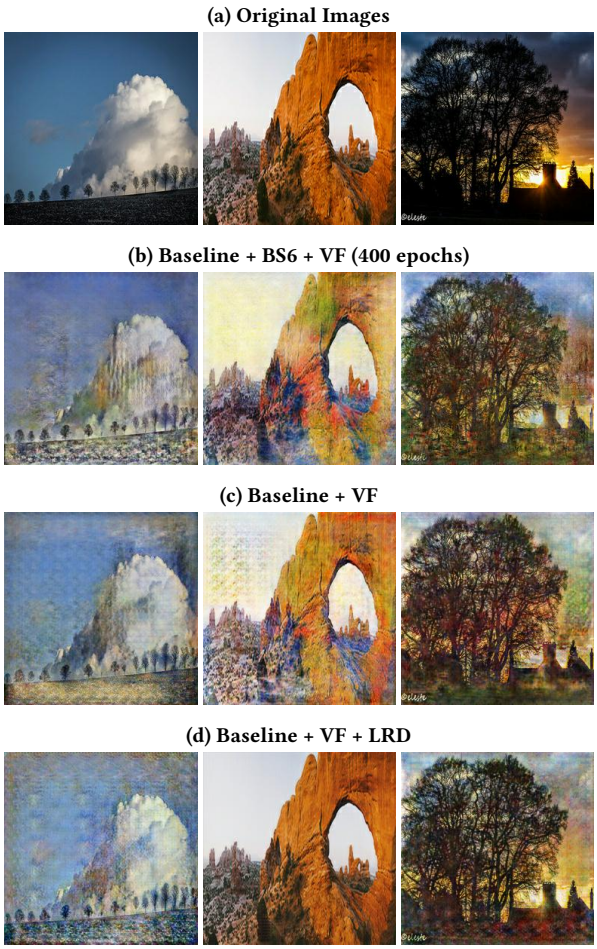

Figure 4: Comparison of original images and the generated output images for our top-3 best performing models.