Picture Monetization

Photo-to-Monet Painting Translation using a CycleGAN

Assignment 2 – EE4685

Joakim Colpier, Jeroen Verweij

1 Introduction

Although the concept of "intelligence" is very ill defined, one readily accepted component of this, which differentiates humans not only from machines but even among ourselves, is *creativity*. By creativity we refer to the ability to produce content that is novel, while still being contextually appropriate. The labeling of AI as "intelligent" and to what extent is a debated topic. However, recent developments in *generative AI models* have further complicated the discussion, as AI models like ChatGPT and DALL-E are now able to create never-before-seen texts and images. One of these methods, capable of generating very convincing new data, is called *generative adversarial network* or GAN as we shall henceforth refer to them. In this report we will implement and evaluate a model that consists of two GANs, called a CycleGAN, which enables image to image translation. One of the benefits of this unsupervised approach is its ability to perform image translation without the necessity of pairwise training data. This modality has a wide range of applications, for example in biomedical imaging [2]. By using CycleGANs as a segmentation algorithm, the workload of medical professionals can be reduced by highlighting regions of interest in relevant image modalities. An example is shown in figure 1 .

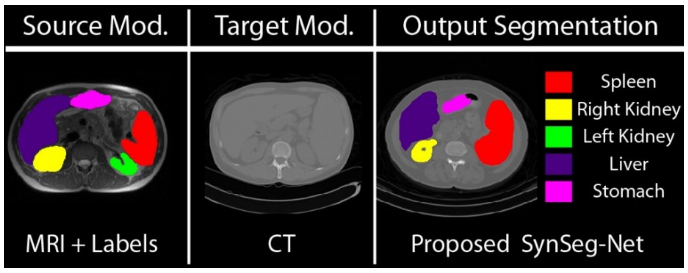


Figure 1: Example of a segmentation network able to transform the labels from a MRI image to a CT image [8]

The goal of this project is to implement a CycleGAN using two distributions: pictures and Monet paintings. We will explain both a non-Bayesian and a Bayesian approach and study their differences, but in practice only implement the former. The code for this project can be found through the following GitHub link.

2 Generative Adversarial Networks

A GAN is a type of implicit probabilistic model which learns a data distribution p, based on a collection of samples. In figure 2 the architecture is shown of a GAN that is trained to generate Monet paintings, in which case the samples of p are real Monet paintings. A GAN consists of two deep neural networks: a generator and a discriminator [12]. The input of the generator consist of noise samples from a distribution $p(\mathbf{n})$. The noise sample is fed through the network to create a fake Monet $\mathbf{x} = G(\mathbf{n}, \boldsymbol{\theta_g})$, where $\boldsymbol{\theta_g}$ is a vector containing the trainable parameters of the network. The discriminator functions as a classifier that takes a sample as an input and returns the probability $y = D(\mathbf{x}, \boldsymbol{\theta_d})$ of the input $\mathbf{x}$ being a real Monet. $\boldsymbol{\theta_d}$ is a vector containing the trainable parameters of the discriminator. The training is an iterative process where the generator tries to fool the discriminator by creating as realistic Monet's as possible, while the discriminator tries to accurately determine which samples originates from the dateset and which are created by the generator. In that sense the components are competing which each other, hence the name "adversarial".

2.1 CycleGAN

A variant of a GAN specializing in conversion between two domains, as opposed to a one-way conversion, is called *CycleGAN*. The general architecture considering the transformation from a real image (domain A) to a Monet (domain B) is shown in figure

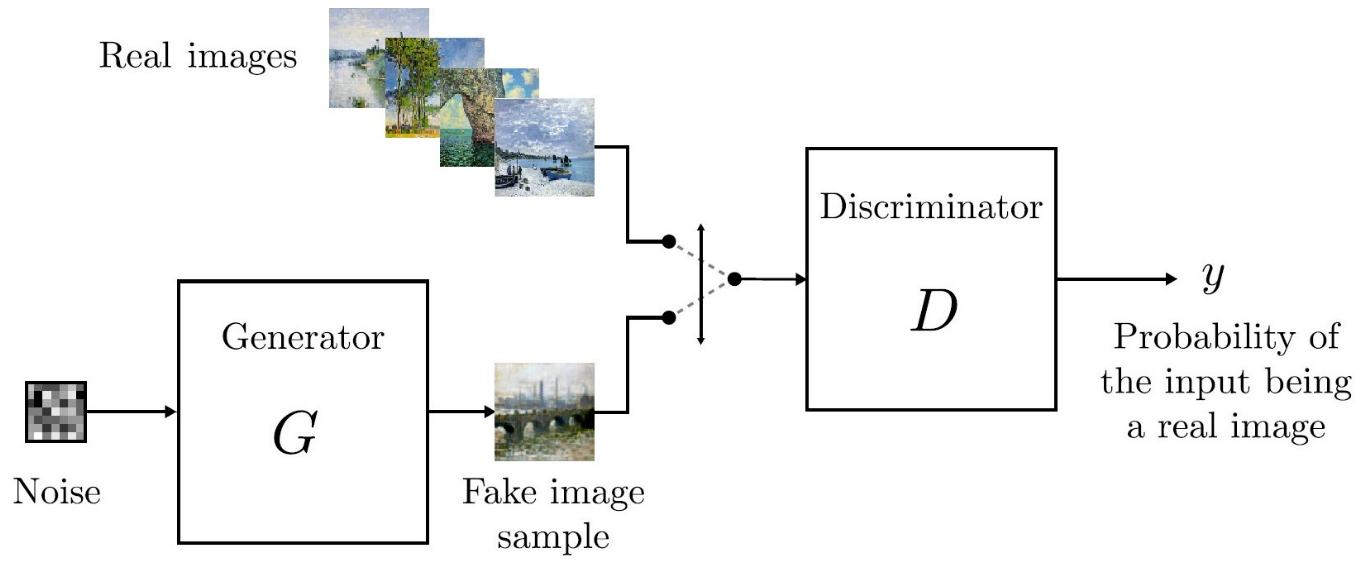


Figure 2: Architecture of a GAN trained to generate Monet paintings, adapted from [12].

3 . The transformation from Monet to photo follows the same architecture, where A and B are interchanged. Compared to the GAN as described in section 2 , the input to both generators isn't noise but rather consists of samples from image distributions. We also introduce an additional set of models: a generator G_{BA} performing the inverse operation and its corresponding discriminator. This introduction of another generator is necessary to produce a sensible translation. In fact, the generator's objective is ambiguous. Its goal is to fool the discriminator by generating an output that resembles a sample from the real image distribution. However, there are infinitely many translations that produce the desired distribution [14]. The generated sample however must be an image that largely resembles the original save from some features characterizing the target distribution. Furthermore, this method suffers from mode collapse, where only a few samples are learned to fool the discriminator, halting learning for both models since they are interdependent in each other's cost functions (as will be seen later in the paper) [14]. It is in order to solve these issues that a second generator is introduced. This generator performs the inverse mapping back to the original domain. In this way we solve the ambiguity between both domains, as both mappings should be bijective to be invertible. To ensure the consistency between the original domain and the reconstruction, both image are compared and assigned a so called *cycle constancy loss*. In addition to this, an *identity loss* is also introduced, ensuring that the generator does not alter an image that is sampled from the source distribution. The loss functions involved in the training process are discussed in more detail in section 4 .

3 Architecture of CycleGAN

In this section, the architecture of the CycleGAN is discussed. Firstly, the architecture of the generator and discriminator will be explored in more detail. Subsequently, we define an important component of these models called the residual block.

3.1 Discriminator

The input of the discriminator is a three-channel 256×256 image, where each channel represents the red, green, and blue contributions. The output consist of one 30×30 channel. The closer the values in this channel are to 1, the higher the likelihood is that the input is a real image according to the discriminator. The same architecture is used as described by Zhu et al. [14]. The network consist of six convolutional layers, including an initial layer, four descaling layers, having 64, 128, 256 and 512 features respectively, and a one channel output layer. The gradual increase in the amount of channels allows the network to account for more complex features. A constant kernel size of 4×4 is used across all layers. For the first three descaling layers a stride of 2 is used. The last two layers are created using a stride of 1. A representation of this architecture is shown in figure 4 .

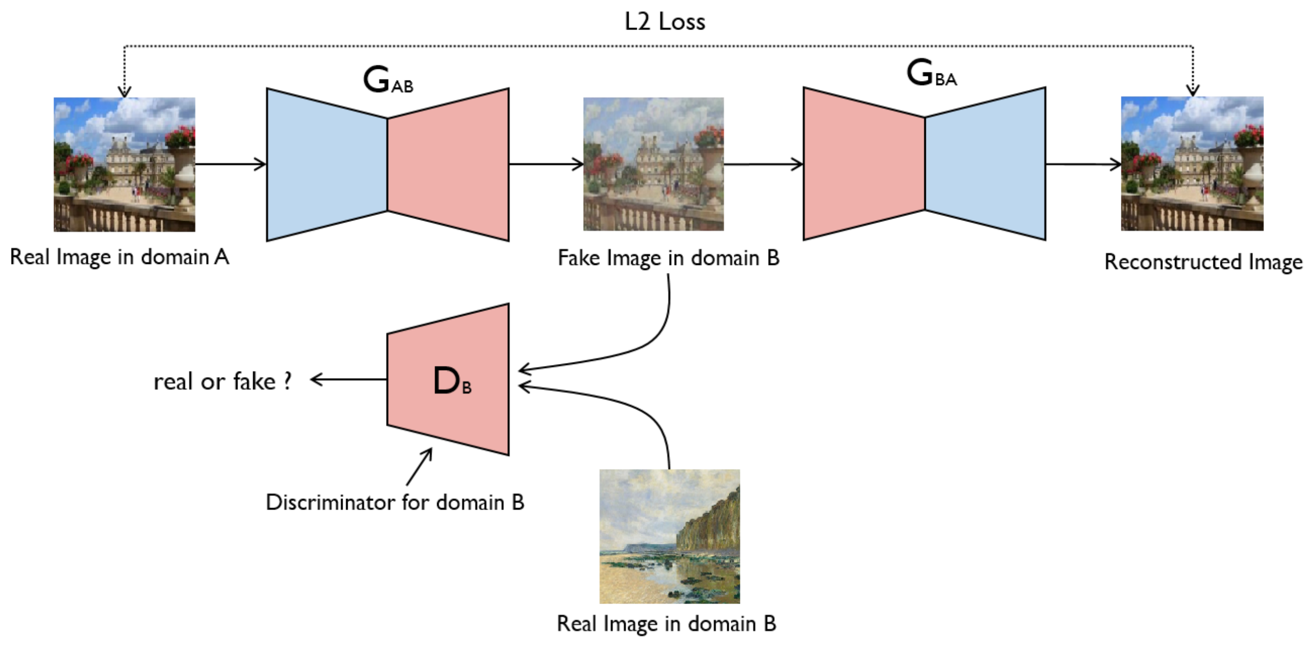


Figure 3: Architecture of cycle GAN adapted from[3] with [14]

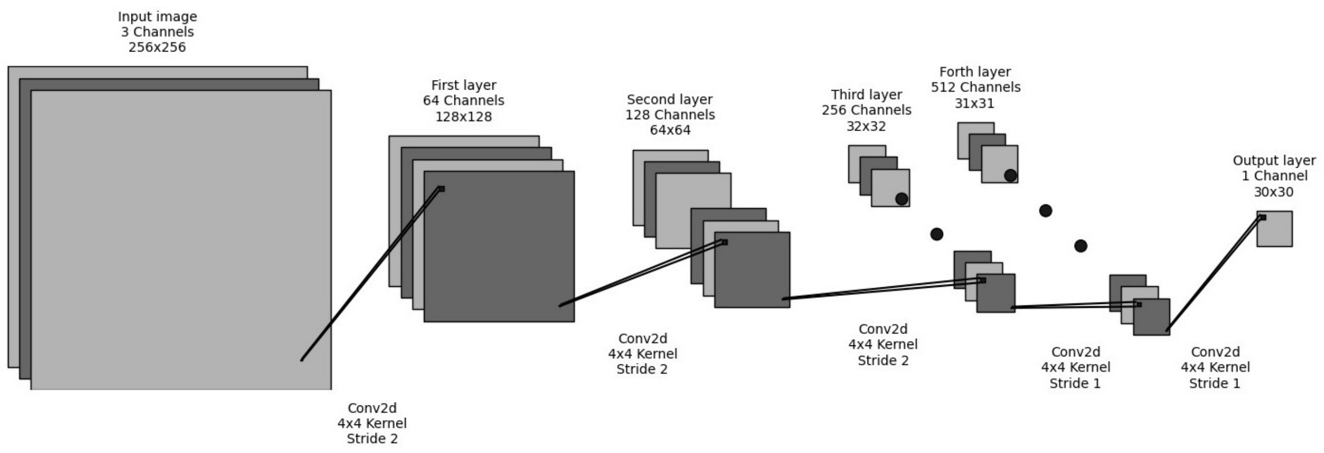


Figure 4: Graphical visualization of the discriminator architecture, created using [5].

3.2 Generator

As for the discriminator, the generator also receives a 3 channel 256×256 image as the input. It returns an image with the same dimensions as the ouput. The generator is a deep neural network consisting of three main components: an encoder, residual blocks and a decoder [3]. The first part of the network, consisting of three convolutional layers, creates a lower dimensional feature representation of the image consisting of 256 features of size 64×64. The central portion of the network can be considered responsible for transforming the features from the input domain to the output domain. It consist of 9 residual blocks. The purpose and the architecture of the residual blocks in this component are further explained in section 6 . Finally, three transpose convolutional layers are applied to upsample the compressed version of the image to the original input size. The architecture again follows the one described by Zhu et al. [14] and is visualized in figure 5 .

3.3 Residual block

It is shown that the depth of networks for image related applications has a large impact on performance, favoring deeper networks [4]. There is, however, a problem with respect to convergence of these deeper networks, which is called the degradation problem as explained by [4]. This problem arises in part from some gradients being zero, which affects all subsequent gradients

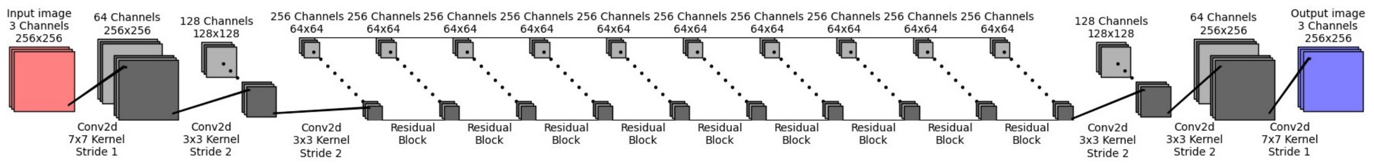


Figure 5: Graphical visualization of the generator architecture, created using [5]

in which the former is used (due to the chain rule), hence leaving that part of the network unable to learn.

This problem can be tackled by dividing the deep network into smaller residual blocks [4], which is also done in the architecture of the generator shown in figure 5 . Instead of trying to learn a feature map $\mathcal{H}(\mathbf{x})$ from the input to the output shown in figure 6 , one can simplify the training by forwarding the input to the output. In this way the network only has to learn the residual $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$ between the two feature domains instead of coming up with a complete mapping by itself. For image-to-image translation this approach makes sense, as the output of the generator is closely related to the input. In addition to this, we effectively add 1 to the gradient of the residual blocks, alleviating the risk for vanishing gradients. In order to perform the addition as shown in figure 6 , the output should have the same dimension as the input. This is ensured by the fact that the convolutional layers both utilize a stride and padding equal to one.

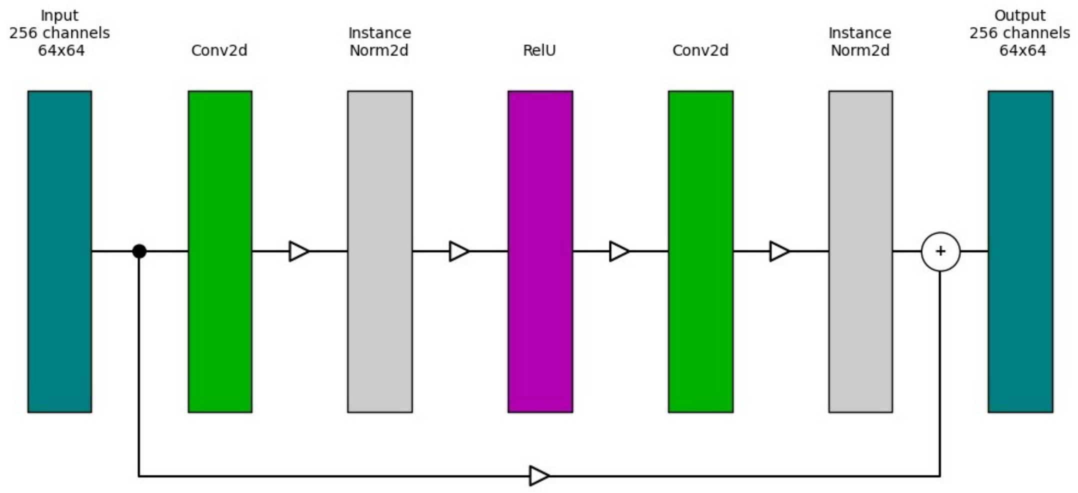


Figure 6: Flowchart of inidividual residual block, created using [5]

4 Training

GANs learn not by approximating distributions explicitly, but rather by developing procedures to generate data from the distributions implicitly. For CycleGANs, this is done by training the generators and discriminators in such a way that they both play a role in each other's loss, from which the parameters are changed. Intuitively, this means that the generator is trained to "fool" the discriminator, while the latter is trained to spot the real from the generated. Mathematically speaking, the losses L_D and L_G for the discriminator D and generator G are the following.

$$L_D = \frac{mse(\mathbf{1} - D_m(\mathbf{x_m})) + mse(D_m(G_m(\mathbf{x_p})))}{2} + \frac{mse(\mathbf{1} - D_p(\mathbf{x_p})) + mse(D_p(G_p(\mathbf{x_m})))}{2} \tag{1}$$

Here we use the notation that $\mathbf{x_a}$ represents a sample from the dataset a, D_a the discriminator taking inputs from dataset a and G_a the generator that replicates the dataset a (m for Monet or p for pictures). $\mathbf{1} \in R^{30\times30}$ represents the matrix equal to the output channel size of the discriminator consisting of 1s. The discriminator is thus improved such that it classifies real and fake data as such, where the fake data comes from the generator. The loss for the latter is defined as follows.

$$\begin{aligned} L_G = &\underbrace{mse(\mathbf{1} - D_p(G_p(\mathbf{x}_m))) + mse(\mathbf{1} - D_m(G_m(\mathbf{x}_p)))}_{\text{adversarial loss}} \\ &+ \underbrace{\lambda_C(||G_p(G_m(\mathbf{x}_p)) - \mathbf{x}_p||_{L_1} + ||G_m(G_p(\mathbf{x}_m)) - \mathbf{x}_m||_{L_1})}_{\text{Cycle loss}} \\ &+ \underbrace{\lambda_I(||G_m(\mathbf{x}_m) - \mathbf{x}_m||_{L_1} + ||G_p(\mathbf{x}_p) - \mathbf{x}_p||_{L_1})}_{\text{Identity loss}} \end{aligned} \tag{2}$$

The loss for the generator function is more complicated and consists of three parts: the adversarial, cycle, and identity loss. The goal of the former is to train the CycleGAN to "fool" the discriminator, and by extension to move more to the distribution of the data. Had this not been a part of the total loss, the generator and discriminator would probably have converged much faster in a sort of middle ground were they both get perfect scores on their loss functions, without actually producing anything sensible. For this reason, no weight has been placed before the adversarial loss, which should never be zero (scaling this in proportion to the other losses is equivalent with scaling the other two weights).

As for the other two losses, they look similar and serve similar purposes. The cycle loss examines the L^1 norm of an image that has been converted to the other distribution and then back minus the unaltered version, whereas the identity loss examines the L^1 norm of the difference between an image converted to its own distribution with its unaltered version. The idea is that we want the former to train the two generators to find the same features to change (and only change those), as well as changing them to something realistic, whereas the latter strengthens the generators' ability to identify what shouldn't be changed with an image and forbids the eventuality of the two generators learning to work together without actually producing what we want. For painting-to-photo translation in particular, the identity loss is helpful to preserve color composition [14]

Together with the adversarial loss, this leads to more realistic images where only relevant parts are changed. The weights λ_C and λ_I are hyperparameters dictating the relative importance of each loss, which we set to 10 and 5 respectively to match those of Zhu et al. [14].

The optimization was achieved using SGD, where each step consisted of first optimizing the discriminator and then the generator as described above. This was performed for every datapoint, making up one epoch. 10 epochs were used in total, a number chosen in part due to the computationally heavy nature of the task (around 50 hours).

4.1 Bayesian approach

The Bayesian approach to GANs that we will outline is the one presented by Saatchi & Gordon Wilson [11]. Note that their implementation concerned GANs and not CycleGANs like we are interested in. As we have seen in the above sections, the difference between the models is small and we can easily adapt the implementation for CycleGANs. However, this means that we do not know whether this approach is the most adapted to CycleGANs and is something to bear in mind in the rest of the paper.

The implementation consisted of transforming random noise $z \sim p(z)$ through the generator $G(z; \theta_g)$ parametrized by θ_g that would fit the data distribution, as well as a discriminator $D(x; \theta_d)$ parametrized by θ_d. θ_g and θ_d are random variables, whose posteriors can be iteratively found by sampling $z^{(i)}$, $i = 1, ..., n_g$ and computing

$$p(\theta_g|z, \theta_d) \propto \left(\prod_{i=1}^{n_g} D(G(z^{(i)}; \theta_g); \theta_d) \right) p(\theta_g|\alpha_g)$$
$$p(\theta_d|z, x, \theta_g) \propto \prod_{i=1}^{n_d} D(x^{(i)}; \theta_d) \times \prod_{i=1}^{n_g} (1 - D(G(z^{(i)}; \theta_g); \theta_d)) \times p(\theta_d|\alpha_d)$$

where $x^{(i)}$, $i = 1, ..., n_d$ are the data points and $p(\theta_g|\alpha_g), p(\theta_d|\alpha_d)$ are the parameters' priors. Note again the lack of identity and cycle loss as well as z being noise rather than samples from a target distribution since we are here referring to GANs.

Note that in this formulation, $p(z)$ represents our source distribution. Classical GANs as described in earlier subsections are thus Bayesian GANs with *uniform priors* and choosing the parameters as the *maximum of their posteriors* as opposed to sampling over the posterior distribution. Hence, Bayesian GANs are a generalization of classical GANs. Note that while this was not explicitly described by Saatchi & Gordon Wilson, the prior distribution should most likely be chosen such as to ease computations as much as possible, for example normally distributed.

5 Evaluation

In the following subsections, the metrics used to evaluate both the quality of the generated images and the performance of the discriminator-generator pair are presented. In addition, data partition between the training and validation is discussed.

5.1 Data division

The data used consisted of 300 images of Monet paintings and 7038 pictures. The data had a resolution of 256×256 pixels and was downloaded from kaggle [9]. To determine the performance of the CycleGAN, 75% of the given data was used for training and 25% for evaluation. More specifically, 225 Monets and 5278 photos were used for training, while 75 Monets and 1760 photos were used for validation. This division was chosen in order to use as many images for training as possible, while still giving a reasonably representative validation distribution.

5.2 Assessing image quality

There are several ways to assess the quality of the generated images. This consists of both qualitative and quantitative approaches. One qualitative approach is manual inspection of the images. However, this is not feasible if a large number of images are considered. Furthermore, it is subjective and biased. Therefore, we will consider two quantitative evaluation methods to determine the performance of the network and when to stop training: Fréchet Inception Distance (FID) and CMMD, using Clip embeddings in combination with the Maximum Mean Discrepancy distance.

The Fréchet Inception Distance (FID) is a famous quantitative evaluation method that assesses the similarity between image distributions [1]. To do this, a smaller higher-order feature representation is created by traversing the images through a pretrained InceptionV3 network. The penultimate pooling layer of this model serves as the feature representation with dimensionality of 2048 [10]. This feature vector is assumed to follow a multivariate Gaussian distribution. The difference between the real image distribution $\mathcal{N}(\mu, \Sigma)$ and the generated image distribution $\mathcal{N}(\mu_g, \Sigma_g)$ is calculated by the Fréchet distance, leading to the FID score [7]:

$$FID = ||\mu - \mu_g||_2^2 + tr(\Sigma + \Sigma_g - 2(\Sigma\Sigma_g)^{1/2}) \tag{3}$$

However, there are some limitations to this score, such as the Gaussian assumption [1], which can lead to completely incorrect results when not applicable [10]. In addition to that, Jayasumana et al. shows that this assumption is in general incorrect [10]. Furthermore, the quality and the reliability depend on the correctness of the mean and variance estimates, for which sufficient data is needed. More specifically, the estimation of the covariance matrix, which has dimensions 2048×2048, needs a large sample size to prevent large errors [10]. Therefore, FID is not a sample efficient method.

CMMD solves these issues, as it is sample efficient, providing consistent estimates even for small image sets [10]. Furthermore, it does not make any assumptions about the distribution and the CLIP model used is trained on 400 million images, compared to the inception model which is trained on 2 million ImageNet images, making it a more qualified option [10]. We will therefore use this measure as a benchmark for convergence.

The MMD distance between two probability distributions P and Q in $\mathbb{R}^d$ $\text{MMD}^2(P, Q)$ is calculated as follows [10]. Here, k denotes a positive definite kernel.

$$\text{MMD}^2(P, Q) = \mathbb{E}_{x,x' \sim P}[k(x, x')] + \mathbb{E}_{y,y' \sim Q}[k(y, y')] - 2\mathbb{E}_{x \sim P, y \sim Q}[k(x, y)] \tag{4}$$

An unbiased estimator is given by the following formula [10].

$$\widehat{\text{dist}}^2_{\text{MMD}}(X, Y) = \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{\substack{j=1 \\ j \neq i}}^{m} k(x_i, x_j) + \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} k(y_i, y_j) - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(x_i, y_j). \tag{5}$$

Here, $X = \{x_1, ..., x_m\}$ and $Y = \{y_1, ..., y_m\}$ are sets of samples from distributions P and Q respectively. The following kernel is used, for $\sigma = 10$.

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \tag{6}$$

The CMMD is calculated using the code provided by the authors of the paper [6]. After every epoch, the validation data is forwarded through the CycleGAN with the current parameters to create 75 photos and 1760 Monet's. This data is then compared with the original validation data to yield the CMMD for both the Monet and Photo generator.

5.3 Model performance

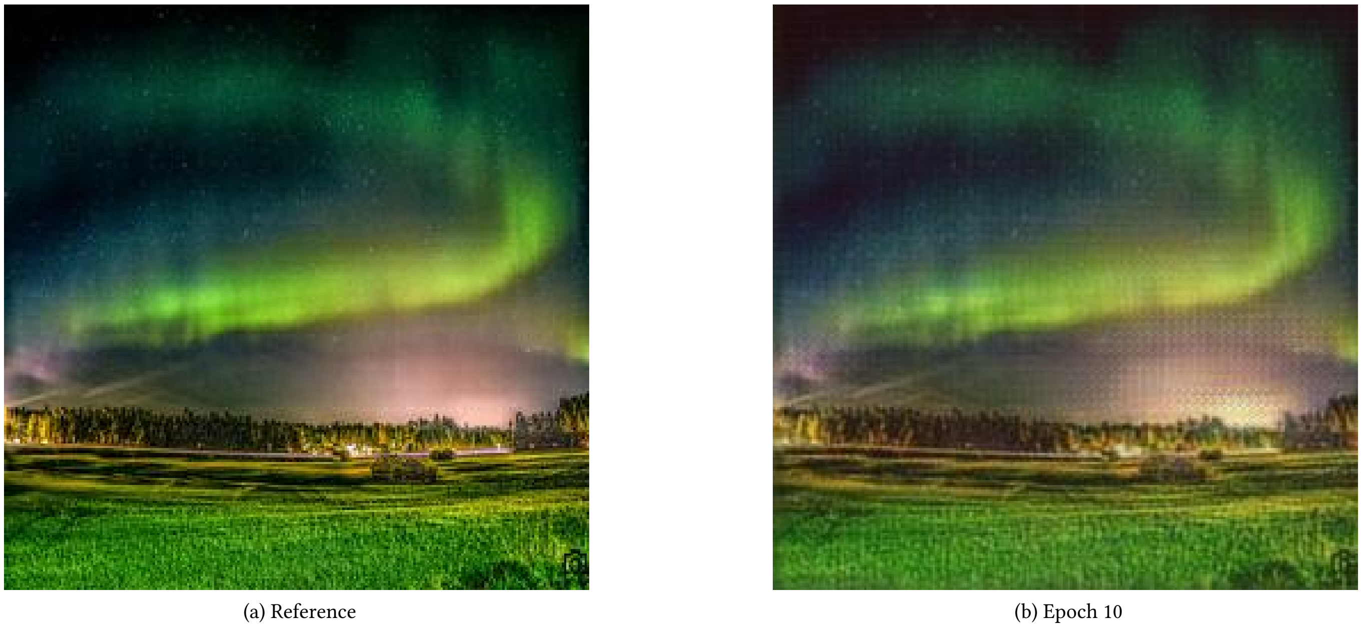
After each training epoch, the performance of both the generators and the discriminators is evaluated. This is also assessed based on the validation data set. For the whole validation set, the average discriminator loss is calculated according to equation 1 . Same is done for the generators, using equation 2 .

6 Results

In this section, we present the results of our CycleGAN implementation. The average losses are shown for both the generator and the discriminator across 10 epochs. In addition, the CMMD scores for both the generated pictures and Monets are presented. By means of visual inspection, the average and best images are shown after 10 epochs. The evolution of these images across the different epochs is shown in the appendix.

6.1 Monets

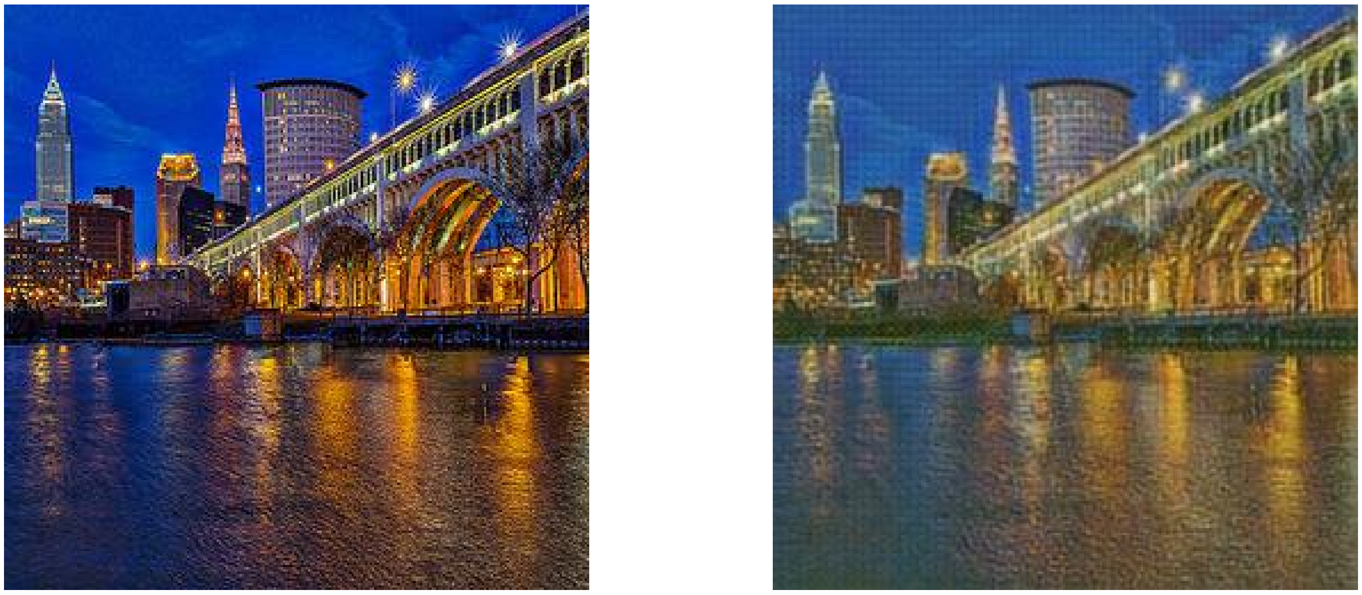
6.1.1 Average performance



(a) Reference (b) Epoch 10

Figure 7: Reference photo and corresponding Monet after 10 epochs

6.1.2 Best performance

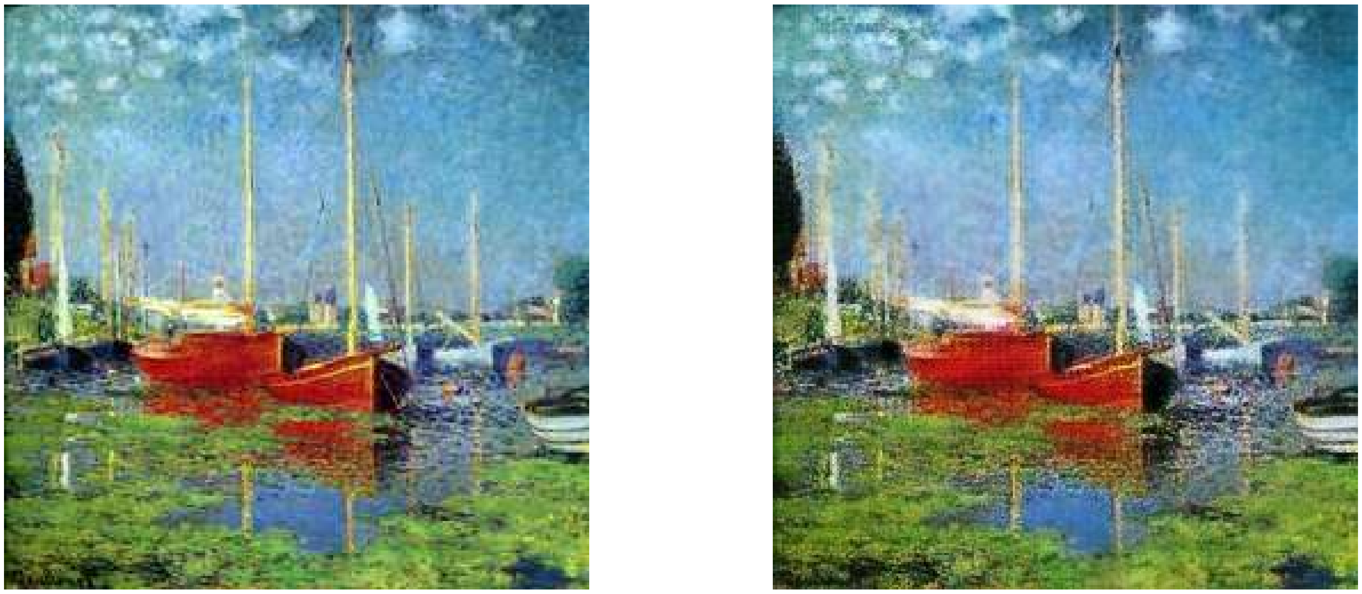


(a) Reference (b) Epoch 10

Figure 8: Reference photo and corresponding Monet after 10 epochs

6.2 Photos

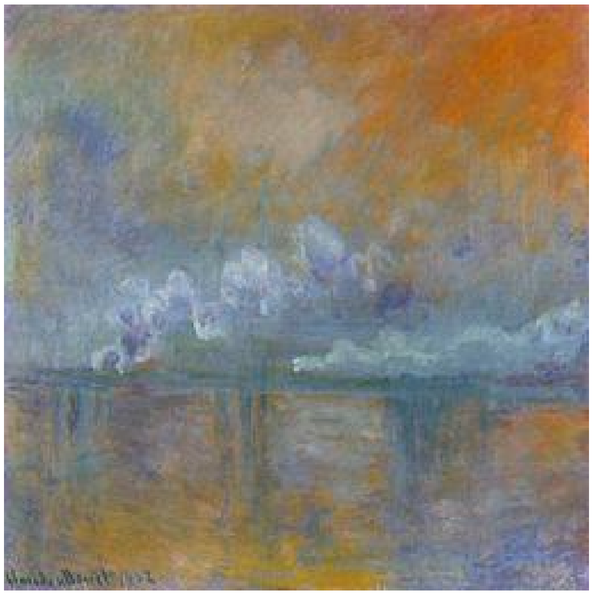
6.2.1 Average performance



(a) Reference (b) Epoch 10

Figure 9: Reference Monet and corresponding photo after 10 epochs

6.2.2 Best performance


(a) Reference

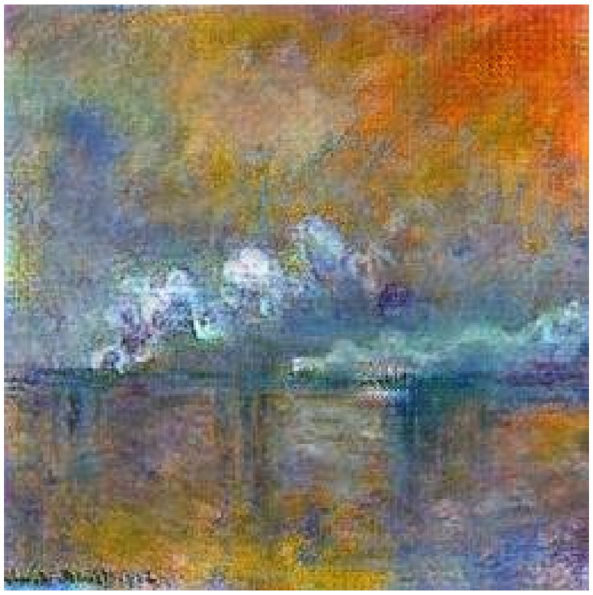
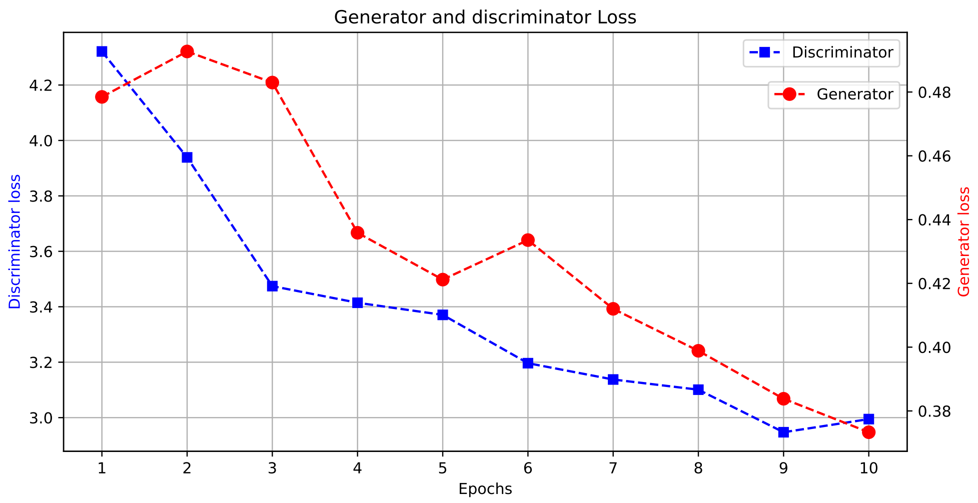
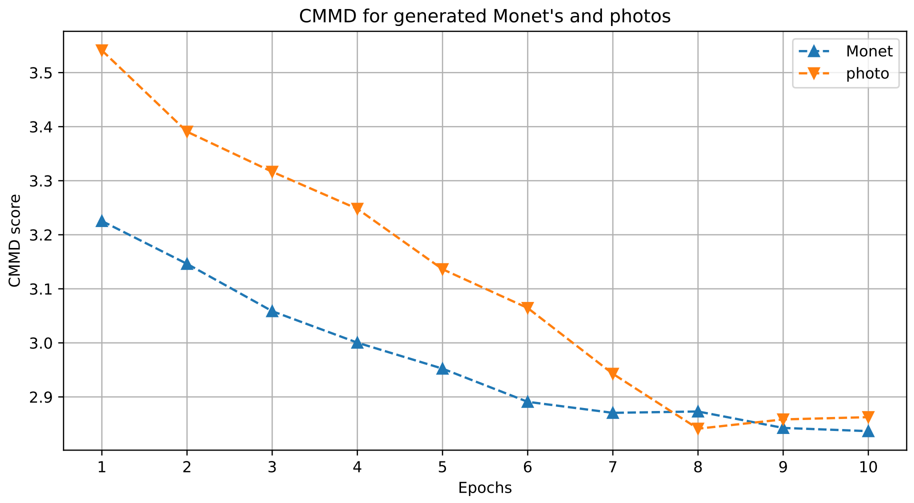

(b) Epoch 10

Figure 10: Reference Monet and corresponding photo after 10 epochs

6.3 Model performance



(a) Losses of the generator and the discriminator



(b) CMMD score

Figure 11: Quantitative performance of the network

7 Discussion

In this section, we will analyze the results presented in section 6 , compare the implemented non-Bayesian approach with a Bayesian implementation of a CycleGAN, and finally compare our results with the literature.

7.1 Results discussion and literature comparison

We can see from figure 11 that both the losses and the CMMD score go down for higher epochs, meaning that the network is learning. While the CMMD is quite stably facing downwards, we see that the generator and discriminator jump around much more. This is to be expected, however, since a better generator will give a higher loss to the same discriminator than a worse one, meaning that the curb will not be stable. We also see from the figures in the appendix A that the generator has changed since the first iteration. Indeed, we see that it has correctly identified saturation and sharpness as a factor separating the two distributions. This can clearly be seen from figure 10 where the clouds are made more contrastive against the background and the colors pop more, whereas in 8 the opposite is happening. While less evident from the given images, the model also seems to have identified the characteristic impressionistic style of the Monet paintings (the broad brush strokes). This can for example be seen in the sky in figure 9 and the water in figure 8 .

While the performance indicators are all pointing downwards, we see that they are still relatively high compared to the first epoch (a perfect score being 0). Furthermore, they seem to stagnate in the last few epochs. In fact, while the model indeed seems to have identified some characteristics of both distributions, we can clearly see what distribution the images were previously. We see from Zhu et al. [14] that it is possible to generate very convincing images using CycleGANs. In comparison to theirs, our results seem quite underwhelming. While we do manage to generate images resembling the original image, we can see from the figure in section 6 that they do not really follow the desired distribution. Several reasons may lie behind this. One reason is that the amount of data is very low, especially from the Monet dataset. However, we see from the CMMD score in figure 11 that while there is a difference between the models connected to the two datasets, it is not substantial. In fact, they both seem to converge to around the same values by the end. Additional data might still improve the quality of the model, however, since all constituent parts are linked (few Monet data could affect the models based on the photo distribution as well). Another reason could be the low number of epochs ran. Since we see a very clear downward trend, we could suppose that this would continue at least for a few more epochs, giving better results since we are still far from a local minimum. Again, however, the CMMD seems to stagnate around the end. This indicates that there may be something else blocking the learning.

One last reason explaining the performance of the results is the fact that either the generator or the discriminator is much better than the other. Since the loss for one depends on the other, a performance gap between the two could lead to slower progression. In fact, looking at the losses in figure 11 , we see that the discriminator racks much higher values than the generator. Furthermore, as we have discussed earlier, GANs are famously prone to mode collapse. This would indeed give us a high local minimum compared to the global one. It also explains why the model hasn't identified more key differences between the distributions – if the discriminator cannot make out the difference between the real and fake data, the generator cannot improve, and neither can the discriminator, hence why we are stalling.

7.2 Comparison with the Bayesian network

As argued by Saatchi & Gordon Wilson [11] and You et al. [13], there are two substantial problems with the classical approach to GANs. Firstly is the "mode collapse", i.e. when the generator simply learns a few examples to trick the discriminator. This makes the discriminator unable to distinguish the real from the fake and gives it quite meaningless gradients to work on, which in turn limits the generator's potential. Secondly, the generated posterior has a tendency to be overly compact relative to samples from the data distribution [11], meaning that it does not produce as diverse results as we would expect. Both papers claim that these problems are tackled by the implementation of a Bayesian approach, although notably, they only experiment with semi-supervised learning as opposed to supervised learning like we are facing. As discussed earlier, they also only implement the Bayesian approach on regular GANs, and not CycleGANs. The argument for why the Bayesian approach would solve our problems are as follows. Since the generator and discriminator don't take a single maximum but rather sample over a distribution, the output will naturally be more diverse and more resemble the breadth of the desired distribution. As a result, the generator cannot simply learn a few examples to trick the discriminator as it samples randomly from its distribution, meaning we avoid mode collapse and get more efficient training.

As argued for in subsection 7.1 , these are precisely the problems we seem to be facing. On the one hand, we seem to have a discriminator that performs much worse than the generator, and on the other, our outputs do not seem to be very different from the inputs. While we have argued for other solutions to these problems, mainly by letting the code run for more epochs and increasing the input dataset, these are not necessarily feasible in this case and certainly not in general. Computation power will often be a bottleneck and including more data may be difficult, impossible or unethical (e.g. the process of procuring a lot of sensitive information may lead to them not being properly handled, or unethically procured). Alleviating the mentioned problems would thus increase the speed with which the model learns as well as giving more varied results.

8 Conclusion

In this paper, we have implemented a non-Bayesian CycleGAN between photos and images of Monet pictures. We trained this model for 10 epochs, and studied these results. While we do get recognizable output images, they do not really follow the desired distributions and give relatively high losses and CMMD scores. We know from literature that it is possible to achieve better results with this type of CycleGAN, however. Possible reasons behind these results include a low amount of input data (especially of Monet paintings), an insufficient number of epochs, and a discrepancy in the performances of the discriminators and generators. A method from the literature that addresses these problems is the Bayesian approach for CycleGANs. It alleviates the problem with a narrow output distribution and unevenness between discriminator and generator performances, leading to faster convergence and better results.

We thus encourage future research in the subject to study *whether*, and if so *when*, Bayesian CycleGANs truly perform better than non-Bayesian implementations. We have seen for other classical and Bayesian machine learning model pairs (such as AEs and VAEs) that while the former generalizes better, it also gives rise to less exact ("blurry") outcomes, and it would be interesting to see in future research if we can see similar trends for CycleGANs. We also encourage research to take into consideration the very long training times that come with CycleGANs, as this makes it harder to work with. In fact, an area of study would be studying the length of Bayesian CycleGAN's training times as that reflection was absent from the source used. In general, it would be very interesting to see how the training time for CycleGAN's in general could be minimized, as this makes them very hard to work with.

8.1 Ethical discussion

An interesting question parallel to what future research could do, is to reflect over what its consequences could be and by extension, *whether* future research should be conducted at all. In fact, CycleGANs and GANs in general give very realistic output that can be modified relatively easily to fit a wide variety of goals. This means that democratizing its use and improving the technique may open doors to its misuse. On the one hand, being able to swiftly make images change distributions may be very useful, e.g. for self-driving cars not needing costly LIDAR equipment or labeling by hand to extract useful information from the environment. On the other, improvements in the method may put yet another strain on artists or lead to an increase in the ease with which it can be used to trick people, and thus its frequency. For example, it could be used for frauds of various kinds owing to how easy it is to impersonate someone's voice or appearance, or for disinformation. While a full reflection on the topic lies beyond the scope of this paper, we believe that future research in the topic should take these thoughts into consideration.

References

[1] Zeeshan Ahmad et al. "Understanding GANs: fundamentals, variants, training challenges, applications, and open problems". en. In: *Multimed. Tools Appl.* (May 2024).

[2] Junhua Chen et al. "Deep learning based unpaired image-to-image translation applications for medical physics: a systematic review". en. In: *Phys. Med. Biol.* 68.5 (Feb. 2023).

[3] *CycleGAN — kaggle.com.* `https://www.kaggle.com/code/himasha0421/cyclegan/notebook`. [Accessed 25-03-2025].

[4] *Deep Residual Learning for Image Recognition — doi.org.* `https://doi.org/10.48550/arXiv.1512.03385`. [Accessed 25-03-2025].

[5] *GitHub - nhansendev/PyDrawNet: A python utility for plotting neural network (and other) diagrams — github.com.* `https://github.com/nhansendev/PyDrawNet`. [Accessed 25-03-2025].

[6] *GitHub - sayakpaul/cmmd-pytorch: PyTorch implementation of CLIP Maximum Mean Discrepancy (CMMD) for evaluating image generation models. — github.com.* `https://github.com/sayakpaul/cmmd-pytorch`. [Accessed 25-03-2025].

[7] Martin Heusel et al. "GANs trained by a two time-scale update rule converge to a local Nash equilibrium". In: *arXiv [cs.LG]* (2017).

[8] Yuankai Huo et al. "SynSeg-Net: Synthetic segmentation without target modality ground truth". In: *arXiv [cs.CV]* (2018).

[9] *I'm something of a painter myself.* `https://www.kaggle.com/competitions/gan-getting-started/data`. [Accessed 17-03-2025].

[10] Sadeep Jayasumana et al. "Rethinking FID: Towards a better evaluation metric for image generation". In: (2024). eprint: `2401.09603` (cs.CV).

[11] Yunus Saatchi and Andrew G Wilson. "Bayesian GAN". In: *Advances in neural information processing systems.* 2017, pp. 3622–3631.

[12] Sergios Theodoridis. *Machine learning.* en. 2nd ed. San Diego, CA: Academic Press, Mar. 2020.

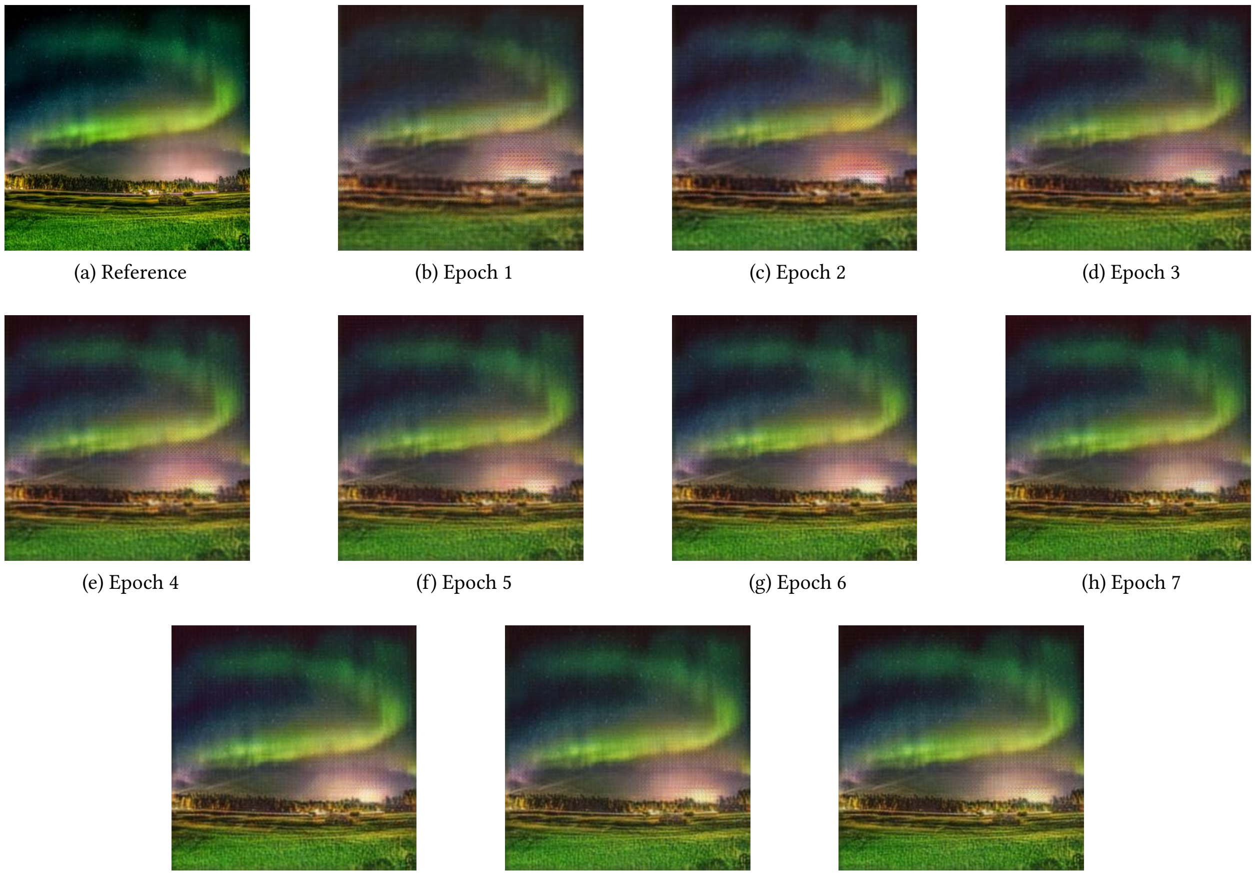
[13] H. You et al. "Bayesian Cycle-Consistent Generative Adversarial Networks via Marginalizing Latent Sampling". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1–15. ISSN: 2162-2388. DOI: `10.1109/TNNLS.2020.3017669`.

[14] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". In: *Computer Vision (ICCV), 2017 IEEE International Conference on.* 2017.

A Appendix

A.1 Monets

A.1.1 Average performance



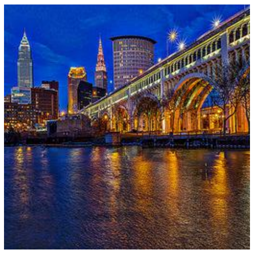
(a) Reference (b) Epoch 1 (c) Epoch 2 (d) Epoch 3

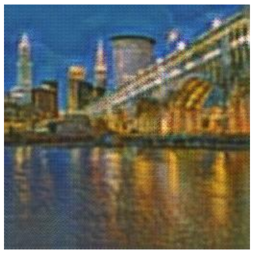
(e) Epoch 4 (f) Epoch 5 (g) Epoch 6 (h) Epoch 7

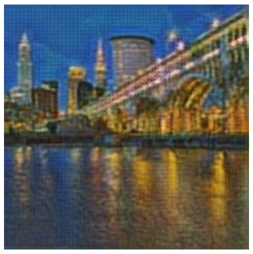
(i) Epoch 8 (j) Epoch 9 (k) Epoch 10

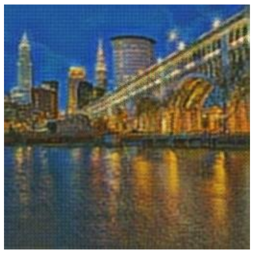
Figure 12: Evolution of fake Monets across epochs

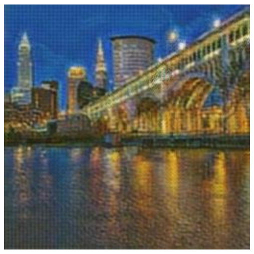
A.1.2 Best performance

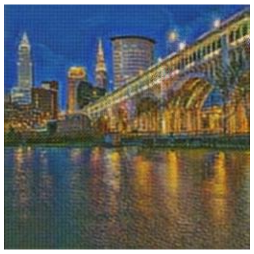

(a) Reference

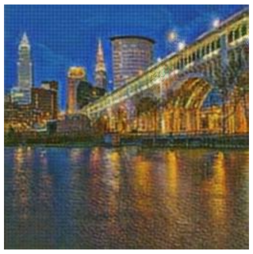

(b) Epoch 1

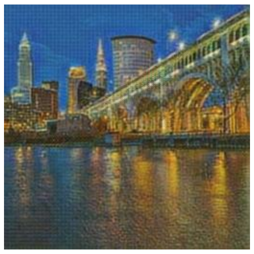

(c) Epoch 2

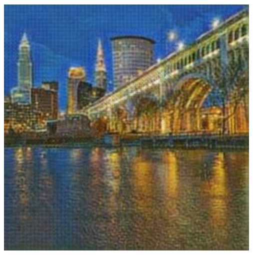

(d) Epoch 3

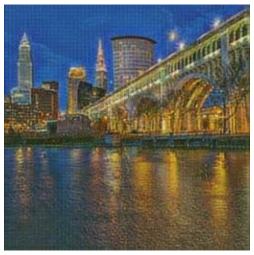

(e) Epoch 4


(f) Epoch 5


(g) Epoch 6


(h) Epoch 7


(i) Epoch 8


(j) Epoch 9

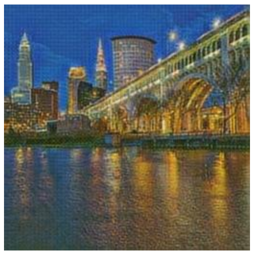
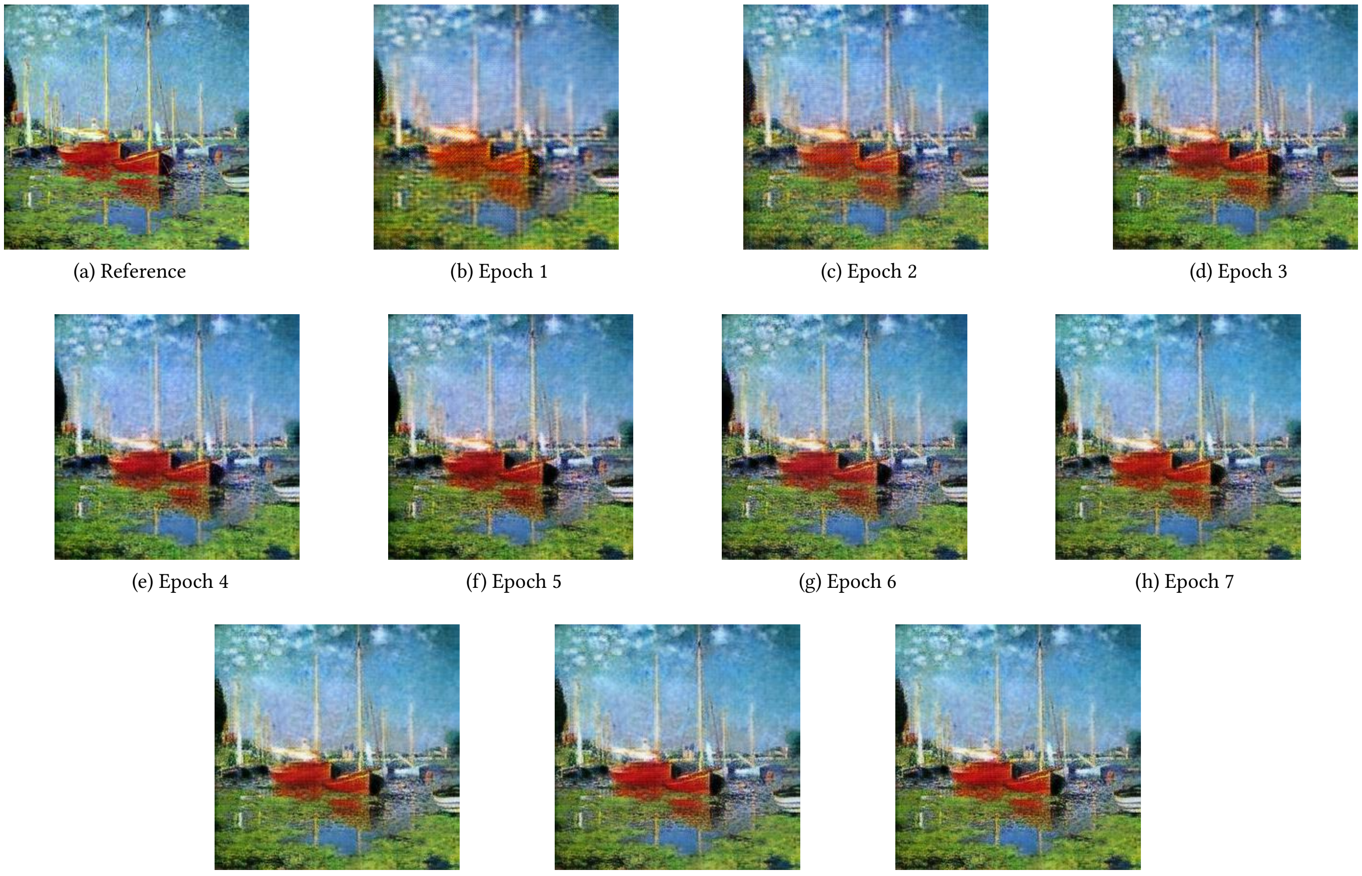

(k) Epoch 10

Figure 13: Evolution of fake Monets across epochs

A.2 Photos

A.2.1 Average performance



(a) Reference (b) Epoch 1 (c) Epoch 2 (d) Epoch 3

(e) Epoch 4 (f) Epoch 5 (g) Epoch 6 (h) Epoch 7

(i) Epoch 8 (j) Epoch 9 (k) Epoch 10

Figure 14: Evolution of fake photos across epochs

A.2.2 Best performance

(a) Reference

(b) Epoch 1

(c) Epoch 2

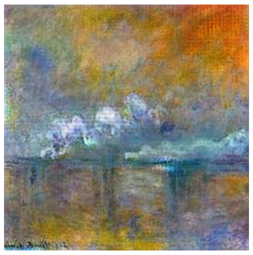
(d) Epoch 3

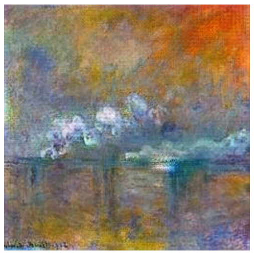
(e) Epoch 4

(f) Epoch 5

(g) Epoch 6

(h) Epoch 7


(i) Epoch 8


(j) Epoch 9

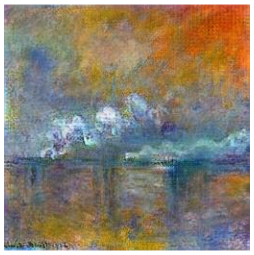

(k) Epoch 10

Figure 15: Evolution of fake photos across epochs