Applied Machine Learning - EEEM068
Human Faces Generation with Diffusion Models

Artem SOKOLOV
UN: 6838385
as05633@surrey.ac.uk

Antoine EDY
UN: 6797948
ae01116@surrey.ac.uk

Pavlos AQUINO-ELLUL
UN: 6833532
pa00768@surrey.ac.uk

Christina HADJIZORZI
UN: 6831502
ch01912@surrey.ac.uk

Abstract

This project explores the field of diffusion models, which have acquired popularity in the domain of Artificial Intelligence thanks to their extraordinary image-generating capacity. This increase in popularity is also due to the interest shown by the general public, notably as a result of the launch of models such as Dall-E [14] or Stable Diffusion [15], which have astonished all their users — the non-AI experts too. While AI models have been impressing scientists for decades, diffusion models - as well as Large Language Models) have made it possible to reach the mainstream, contributing to make artificial intelligence trendy.

Our task focuses on using a diffusion model [1, 19] to generate human faces. Achieving realistic and aesthetically pleasing results is challenging: human faces are highly intricate and nuanced structures, and capturing their complexity is far from simple. Moreover, the space of possible human faces is vast, with numerous variables contributing to facial appearance, including shape, colour, texture, lighting, and expressions. Finally, it is easy to fall into the uncanny valley [3], which refers to the phenomenon where a humanoid appears almost, but not exactly, like a real human, which causes discomfort in observers.

We will see that diffusion models can overcome these challenges and produce results than can easily fool humans.

1. Introduction

The aim of this project is to train a diffusion model on human face dataset, in order to undertake the task of unconditional generation of images of faces. The dataset used will be the CelebA-HQ 256×256 [8], a dataset containing 30,000 high-quality celebrity faces (resampled to 256px in our case, the original one is 1024px). This report will go through our the journey of applying diffusion models for image generation, beginning with an exploration of the theoretical underpinnings of diffusion models during the literature review (section 2) and the main concepts behind diffusion (section 3). The report will then detail the practical implementation in section 4 by first training on a butterfly dataset to grasp the fundamentals (section 4.1). The focus then shifts to the main objective: the generation of human faces using the CelebA-HQ dataset in section 4.2.

Throughout this report, we will investigate the different effects of hyperparameter changes on image generation, and use the Fréchet Inception Distance (FID) score as our primary measure of quality (detailed in section 2.2).

2. Literature Review

Diffusion models can be split into three distinct categories, Denoising Diffusion Probabilistic Models (DDPMs) [6, 12], score-based generative models (SGMs) [18] and stochastic differential equations (SDEs) [17]. Our case falls under the first category, DDPMs, where noise is added progressively to the data and then the model. A commonly used architecture to reverse this process (and generate a meaningful image) is U-net, detailed in section 2.1. The diffusion process consists of two Markov chains, one for the forward pass (noising) and one for the backwards pass (denoising). However, there are different types of generative models, such as Generative Adversarial Networks (GANs) [4] and Variational Autoencoders (VAEs) [9], that we won't study in this report.

2.1. U-net

U-net, as proposed by Ronneberger et al. [16], is a CNN architecture, originally designed for Biomedical Image Segmentation. It has a roughly symmetrical architecture, as

seen in figure 11 (appendix), and does not contain any fully connected layers. To classify border pixels, the image is mirrored within the bounds of the kernel. However, U-net was later used for diffusion [7], as part of the backwards process of denoising the images.

2.2. Fréchet Inception Distance

Initially, the Fréchet distance between multivariate normal distributions was introduced in 1982 [2], as a way to express with a number the difference between two distributions. The original formula can be seen in Equation 1. Later in [5], the Fréchet Inception Distance (FID) was introduced, an adaptation of the original formula, for Generative model evaluation. In the case of generative models, the two distributions are an aggregation of the features of the real images and generated images. Here, the FID informs us on how close the feature distribution of the generated images to the real images (figure 10, appendix). Intuitively, we want the FID to be small, so that the model generates realistic images. The ideal value of the FID score is zero, and we consider a good value being under 100. However, could this entail lack of variance and creativity, or even reproduce images really close to the original ones? It is not the case in practical, because reaching an FID score of zero or very close to zero is extremely hard. Another image-similarity score sometimes used is the Inception Score, which is not as consistent as the FID [5].

$$d^2_{(x,y)} = \|\mu_x - \mu_y\|^2 + \text{tr}\left(\Sigma_x + \Sigma_y - 2(\Sigma_x\Sigma_y)^{1/2}\right) \quad (1)$$

Equation 1. Fréchet distance between two multivariate normal distributions x and y, introduced in [2], where μ_i is the mean and Σ_i^2 the variance of the distribution i.

3. Methodology

In this section, we will go through different aspects of the diffusion model we implemented, from a general understanding of the concept (section 3.1) to more specific details of our implementation that lead to better results (sections 3.4 to 3.6).

3.1. The principle of diffusion

In comparison to previous image-generation methods [4,11] that generated images in one step, the diffusion model denoises a random image step by step, slowly creating a meaningful picture. This process involves two steps. The first one starts from the "goal" image (a real picture) that we noise, in order to generate a completely noisy output: this is the forward process. We can choose the number of steps as well as the distribution parameters of the noise added at each step. The second process is called reverse process, where the model learns how to denoise the image gradually. This process will create coherent and meaningful images from noisy ones.

3.2. The forward process

x_0 is our original image, and we will consider T steps (so x_T is our final, noisy image). We will call q the distribution of the x_0 image. Each step, we add Gaussian noise to the image according to some known variance schedule $0 < \beta_1 < ... < \beta_T < 1$. We show in the appendix, section 6, that we can express the image at any step the following way (with $\alpha_i = 1 - \beta_i$):

$$x_t = \sqrt{\bar{\alpha}_t}x_\mathbf{0} + \sqrt{1 - \bar{\alpha}_t}\epsilon; \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \quad (2)$$

Thanks to equation 2, we can express any step using the original image x_0.

3.3. The reverse process

Now, we want to be able to compute x_{t-1} from x_t to denoise a noisy image a create a meaningful one. We start from a known noisy image, which distribution can be expressed like so:

$$p(x_T) = \mathcal{N}(x_T; 0, \mathbf{I}) \quad (3)$$

Thus, we want to find $p(x_{t-1}|x_t)$, and we will try to estimate it using a neural network. We will use the parametric notation $p_\theta(x_{t-1}|x_t)$, where θ is the network's parameters. As we deal with Gaussian distributions, $p_\theta(x_{t-1}|x_t)$ can be fully described by a mean and a variance, so:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \quad (4)$$

Here in equation 4 we expressed all the variables the model needs to learn.

3.4. Positional embeddings

As the input to the network, we will pass both the noisy image and some kind of information about the time step t which determines the level of noise. To encode this time step, we used the positional embeddings from the Attention Is All You Need [20] publication proposed by Vaswani et al.

$$\begin{cases} PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}}) \\ PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}}) \end{cases} \quad (5)$$

where PE is the positional encoding matrix, pos is the timestep in our case, i is the dimension index of the embedding, and d_{model} is the dimensionality of the embeddings.

3.5. Weight Standardization

Qiao et al. [13] have shown that normalizing the weights of the convolution layers led to better results. We implement this by defining the new convolution weights as following:

$$W_{\text{norm.}} = \frac{W - \mu_W}{\sqrt{\sigma_W^2}} \tag{6}$$

3.6. The schedulers

As explained in 3.2, we add at each step of the forward process a Gaussian noise of known parameters. Practically speaking, the variance of this noise can be scheduled either linearly or sinusoidally.

Linear:

$$\beta_i = \begin{cases} 0 & \text{if } i = 0 \\ \beta_{\text{start}} + \frac{i \cdot (\beta_{\text{end}} - \beta_{\text{start}})}{\text{num}_{\text{timesteps}}} & \text{if } i > 0 \end{cases} \tag{7}$$

where we chose $\beta_{\text{start}} = 1 \cdot 10^{-4}$ and $\beta_{\text{end}} = 0.02$.

Cosine:

Inspired by [12], we define $\bar{\alpha}_t$:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos\left(\frac{t/T + s}{1 + s} \cdot \frac{\pi}{2}\right)^2$$

To go from this definition to variances β_t, we note that $\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$. We then clip β_t to be greater than 0.0001 and smaller than 0.999.

Using a linear scheduler for the variance in diffusion models results in a straightforward, constant-rate noise addition, while a cosine scheduler offers a smoother, more gradual transition, potentially leading to more stable and higher-quality generative performance due to the more nuanced noise scheduling. Experiments will compare the two in section 3.6.

4. Experimentations

4.1. First experiment with butterflies

The initial experiment used a pre-defined butterfly dataset to train the diffusion model. Figure 1 depicts a sample of this dataset.

Figure 1. Sample of the butterfly dataset

For the training part, the hyperparameters that have been used are 64×64 image size, 32 batch size, 50 loss steps, $1 \cdot 10^{-4}$ learning rate, 100 epochs, and 1000 timesteps. Figure 2 illustrates the loss during the training process, which displays minor fluctuations throughout the epochs.

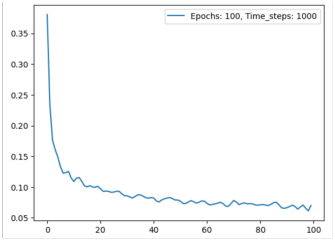


Figure 2. Loss plot for Butterfly dataset

Initially, there is a notable decrease in loss, which then levels off, indicating that it is approaching an optimal solution. The curve's steep drop in the early epochs suggests rapid learning, which stabilizes as the model fine-tunes its parameters. The consistent, low variability in the latter part of the graph shows that the model has achieved a stable state with minor improvements in loss over time.

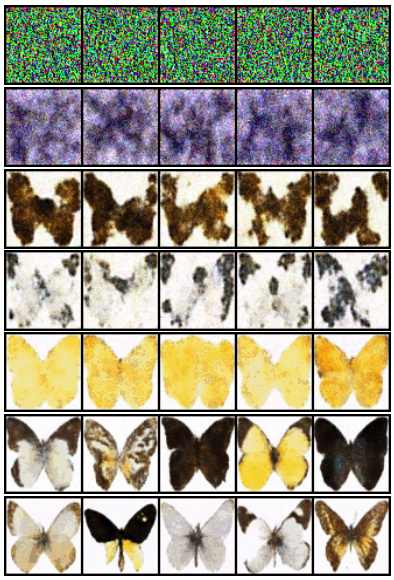


Figure 3. Output through the epochs
(epochs 0, 6, 12, 20, 35, 60, 100)

Figure 3 presents the outcomes from the evaluation function at intervals of every couple of epochs. The final image, corresponding to epoch 100, demonstrates excellent detail and diversity in the generated butterflies.

4.2. Human faces dataset

For the experiments with the face dataset, we utilized 300 images that were not seen during the training phase to generate new images and calculate the FID score (detailed in section 2.2). Initially, the linear scheduler was employed with the following fixed parameters: an image size of 64×64, a batch size of 32, loss steps of 50 and a learning rate of $1 \cdot 10^{-4}$.

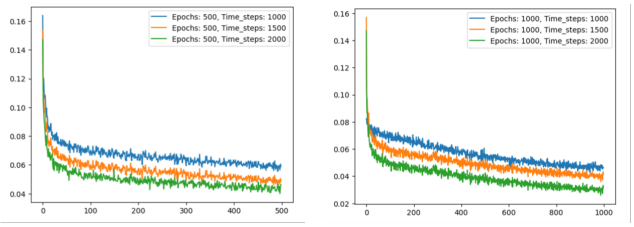
4.2.1 Effect of the number of epochs and timesteps

We ran other experiments to optimize both the number of epochs and the number of timesteps for generating the images.

Epochs	Num. Timesteps	FID score
300	1000	76.93
300	1500	79.14
300	2000	79.63
500	1000	**69.68**
500	1500	88.95
500	2000	78.38
1000	1000	96.80
1000	1500	76.74
1000	2000	73.71

Table 1. FID score using a linear scheduler, varying only the epochs and the number of timesteps (lower is better)

Table 1 depicts the range of hyperparameters used and their corresponding FID scores. The experiment configuration with the lowest FID score (69.68) was achieved with 500 epochs and 1000 timesteps. In contrast, the highest FID score (96.80), indicating poorer image quality, occurred with 1000 epochs and 1000 timesteps. The results from the FID scores were unexpected, as we anticipated lower FID scores with an increased number of epochs and timesteps. However, the loss plots in figure 4 reveal significant fluctuations, particularly noticeable around 1000 epochs with 1000 timesteps in figure 4b, suggesting that the model may overfit. This outcome indicates that simply increasing the number of epochs does not necessarily improve the model's performance, potentially due to its inability to benefit from extended training without corresponding adjustments in model complexity. Enhancing model complexity or introducing regularization strategies might be necessary to better leverage longer training durations without overfitting.



(a) Loss plot for 500 epochs (b) Loss plot for 1000 epochs

Figure 4. Training loss through the epochs

The following image grids (figure 5) display 100 generated images each. In the set generated with 500 epochs, the images exhibit greater diversity and a wider colour range. Conversely, the set generated with 1000 epochs is characterized by a predominance of darker tones and less colour variation.

(a) Generated Images for 500 epochs and 1000 timesteps (b) Generated Images for 1000 epochs and 1000 timesteps

Figure 5. Grids of generated images

4.2.2 Effect of the variance scheduler

Following the above experiment, we used a cosine scheduler (for the variance of the noise added in each step) instead of a linear scheduler for further tests, specifically focusing on the configurations that produced the first and second-best FID results. The FID scores obtained from these experiments, (107.58 for 500 epochs and 1000 timesteps and 80.80 for 1000 epochs and 2000 timesteps), were inferior compared to those obtained using the linear scheduler.

(a) Generated Images using cosine scheduler for 500 epochs and 1000 timesteps (b) Generated Images using cosine scheduler for 1000 epochs and 2000 timesteps

Figure 6. Grids of generated images

Figure 6 illustrates the images generated using the cosine scheduler. Although the results with 1000 epochs and 2000 timesteps were superior to those with fewer epochs and timesteps, they still did not match the performance achieved with the linear scheduler. In fact, a cosine scheduler seems to produce significantly better results when combined with a fixed offset and pixel binning [12], which has not been done in our experiments.
The images from figure 6a, generated with fewer epochs, exhibit less diversity and sharpness compared to figure 6b,

which used more epochs and timesteps. Despite the increased sharpness and clarity in the latter, the overall quality, as measured by the FID score, still lags behind what was achieved with the linear scheduler.

4.2.3 Effect of the learning rate

We end the experiments by tweaking the learning rate of the training process. From now, we used a fixed learning rate value of $1 \cdot 10^{-4}$. We tried $lr \in \{5 \times 10^{-6}, 1 \times 10^{-3}\}$, some extreme values to study the effect of this hyperparameter.

Learning rate	FID score
5×10^{-6}	179.24
1×10^{-4}	**69.68**
1×10^{-3}	337.85

Table 2. FID score comparison for different values of the learning rate — lower is better (timesteps = 1000 and epochs = 500)

As expected, table 2 shows that a learning rate of gives the best results by a large margin. In the case of a large learning rate ($lr = 1 \times 10^{-3}$), the model converges rapidly in the first 200 epochs, but then deconverges, as seen in figure 7. The issue with such a large learning rate is that the optimization process is not precise enough, and the model overtakes the optimal solution at every step, resulting in no convergence.

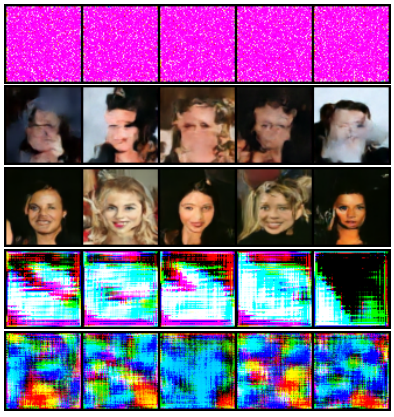


Figure 7. Output through the epochs with $lr = 1 \times 10^{-3}$ *(epochs 0, 35, 210, 300, 500)*

In the case of a small learning rate ($lr = 5 \times 10^{-6}$), the model seems to converge (figure 8) but very slowly. At epoch 500, we achieve worst results than what we did at epoch 200 with a larger learning rate.
We compare the training loss through the epochs in figure 9 for these different learning rates. This plot supports our previous explanations, with the model being unstable with a large learning rate (green line), and too slow with a small learning rate (blue line).

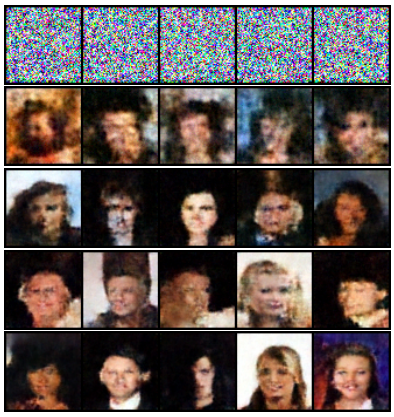


Figure 8. Output through the epochs with $lr = 5 \times 10^{-6}$ *(epochs 0, 35, 210, 300, 500)*

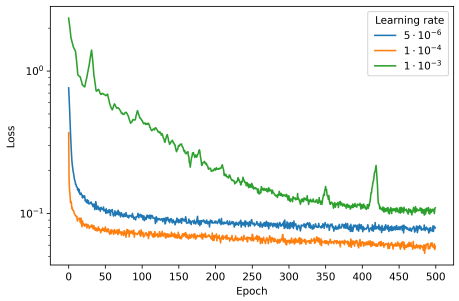


Figure 9. Difference in training loss for different values of the learning rate (y-axis is logarithmic)

5. Conclusion

The project has shown that DDPM can generate realistic and appealing images of butterflies and human faces. By using the CelebA-HQ dataset and experimenting with different settings, we found that these models handle the complexities of human faces well and produce high-quality images. Our results show that the choice of settings, such as the number of training cycles, timesteps, and learning rate, plays a crucial role in the model's success, as measured by the Frechet Inception Distance (FID) score. While the linear scheduler performed best in our tests, future work could explore more advanced scheduling methods and further fine-tuning to enhance the model's performance and image quality. Additionally, conditioning the model on text using Latent Diffusion is another promising direction. Synthetic captions for CelebA-HQ and butterfly datasets could be generated using models like CogVLM [21], allowing the model to be conditioned on text descriptions. Overall, this study highlights the potential of diffusion models in advancing generative modelling, especially for complex data like human faces.

References

[1] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021. 1

[2] DC Dowson and BV666017 Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982. 2

[3] Karam Ghanem and Danilo Bzdok. The uncanny valley: A comprehensive analysis of diffusion models, 2024. 1

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 1, 2

[5] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 2

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. 2

[8] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018. 1

[9] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 1

[10] Tuomas Kynkäänniemi, Tero Karras, Miika Aittala, Timo Aila, and Jaakko Lehtinen. The role of imagenet classes in fr\'echet inception distance. *arXiv preprint arXiv:2203.06026*, 2022. 7

[11] Elman Mansimov, Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Generating images from captions with attention, 2016. 2

[12] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021. 1, 3, 4

[13] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization, 2020. 2

[14] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022. 1

[15] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. 1

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. 1, 7

[17] Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in neural information processing systems*, 34:1415–1428, 2021. 1

[18] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019. 1

[19] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021. 1

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. 2

[21] Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. Cogvlm: Visual expert for pretrained language models, 2023. 5

6. Appendix

A. Calculus

Because we add Gaussian noise to the image each step, we can iteratively express the values of x_t starting from x_0.

$$q(x_t|x_{t-1}) = \mathcal{N}(x; \mu, \sigma^2)$$
$$\text{with } \begin{cases} \mu = \sqrt{1-\beta_T} \cdot x_{t-1} \\ \sigma^2 = \beta_t \cdot \mathbf{I} \end{cases} \tag{8}$$

Then,

$$q(x_{T\to 1} \mid x_0) = \prod_{i=1}^{T} q(x_t|x_{t-1}) \tag{9}$$

And thanks to the properties of the expectation and the variance:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon; \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \tag{10}$$

If we define a new variable α to simplify the expression of x_t:

$$\alpha_t = 1 - \beta_t \quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_t \tag{11}$$

Thus,

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon \tag{12}$$

Or, $E[x_t] = \sqrt{\bar{\alpha}_t} \cdot x_0$ and $Var[x_t] = (1-\bar{\alpha}_t) \cdot \mathbf{I}$, so:

$$x_t = \sqrt{\bar{\alpha}_t}x_{\mathbf{0}} + \sqrt{1-\bar{\alpha}_t}\epsilon \tag{13}$$

B. Appendix Visualizations

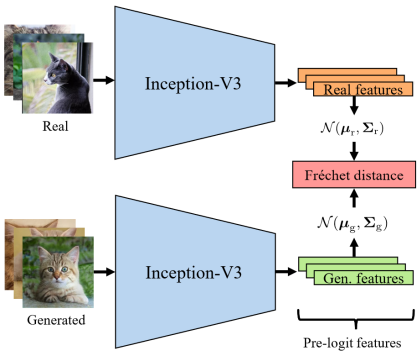


Figure 10. A visual representation of the FID calculation (from Kynkäänniem et al. [10])

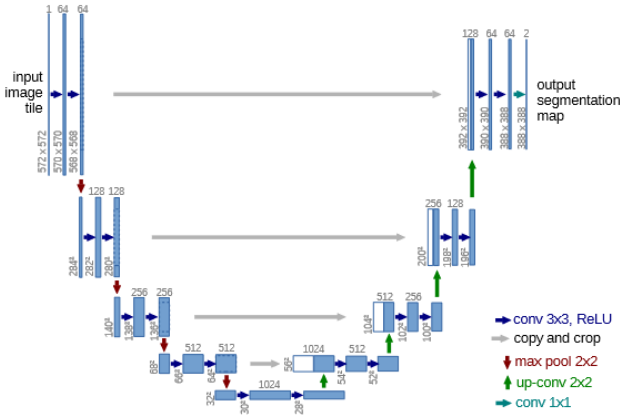
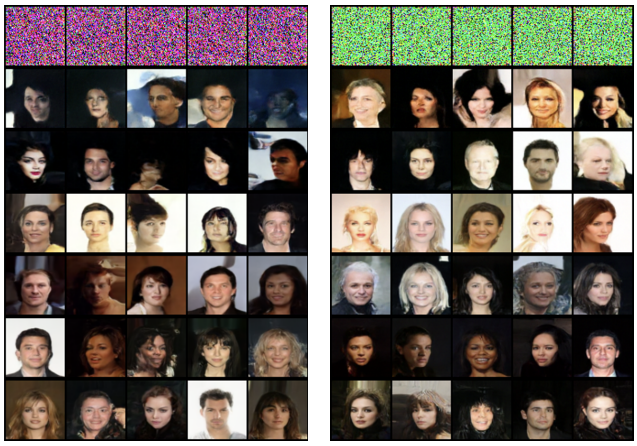


Figure 11. A visual representation of U-net architecture as depicted in [16]

Figure 12. Forward Process to add noise in the Images

Figure 13. Generated Images for 300 epochs and 1000 timesteps



(a) Using linear scheduler for 500 epochs and 1000 timesteps

(b) Using cosine scheduler for 500 epochs and 1000 timesteps

Figure 14. Output through the epochs (epochs 0, 70, 140, 210, 290, 350, 500)

(a) Using linear scheduler for 1000 epochs and 2000 timesteps

(b) Using cosine scheduler for 1000 epochs and 2000 timesteps

Figure 15. Output through the epochs (epochs 0, 110, 220, 330, 440, 550, 660, 770, 880, 1000)