UNIVERSITY OF
CAMBRIDGE

Dmytro Mai

A THOROUGH COMPARISON/ANALYSIS/SUMMARY OF THE PROS & CONS OF THE DIFFUSION MODEL AND GAN

Computer Science Tripos – Part II

King's College

2025

Declaration of originality

I, Dmytro Mai of King's College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this report, I adhered to the Department of Computer Science and Technology AI Policy. I am content for my report to be made available to the students and staff of the University.

Signed Dmytro Mai
Date 18/05/2025

Proforma

Candidate Number:	2204G
Project Title:	A thorough comparison/analysis/summary of the pros & cons of the diffusion model and GAN
Examination:	Computer Science Tripos – Part II, June 2025
Word Count:	11958[1]
Lines of Code:	3490[2]
Project Originator:	Chenliang Zhou
Supervisor:	Chenliang Zhou

Original aims of the project

This project aimed to build two different unconditional generative AI models - Generative Adversarial Network (GAN) and Denoising Diffusion Probabilistic Model (DDPM) and train both of these on three different datasets. The aim was to show the weaknesses of each model by means of quantitative metrics and qualitative analysis of the generated images. Finally, for the extension, Diffusion-GAN was to be implemented and compared to the other two models, with the aim of investigating whether the hybrid model managed to mitigate any of the flaws in GANs and DDPMs.

Work completed

The project was successfully completed, meeting all requirements and extensions. I implemented all three models and recorded the proposed metrics. Additional metrics were introduced to better showcase the phenomena exhibited in each model and for what I argue a better and more fair comparison of the two different classes of models. Extra modifications were made to the DDPM and Diffusion-GAN models which aimed to improve their performance and were made in line with the recent literature. To aid the development, a robust pipeline was implemented for the training and evaluation of the generative AI models.

Special difficulties

None.

[1] Counted using `make wordcount`. Provided in https://www.cl.cam.ac.uk/local/typography/

[2] Counted using `find . \( -name '*.py' -o -name '*.yaml' \) | xargs wc -l .`

Contents

1 Introduction

Overview. With the emergence of generative AI in recent years, deep generative models have managed to produce incredibly high quality samples produced in a variety of different tasks [1], [2], [3]. For the task of image generation, many different models have achieved successful results, namely autoencoder, variational autoencoder (VAE), autoregressive, flow, Generative Adversarial Networks (GANs) and diffusion models. Currently, the majority of models fall under the classes of GAN or diffusion models. The differences between them highlight a key trade-off in generative modeling: diffusion models tend to produce higher quality samples, whereas GANs are more energy efficient. This projects aims to build a diffusion and a GAN model, and formalises their differences by implementing a range of meaningful metrics. Finally, it aims to develop a hybrid - Diffusion-GAN - model to alleviate some of the limitations and bottlenecks encountered in the original two models.

Motivation. Recent advancements in generative AI, such as StyleGAN3 and transformer-based models such as LadaGAN, have made significant improvements in stabilising training and have taken steps towards better capturing image diversity [4], [2], [5]. On the other hand, diffusion class models, such as DDIM and LDM, aimed at improving the sampling speed and convergence speed of the less energy efficient diffusion models [6], [1]. Both sides of research are trying to tackle the generative AI trilemma shown in Figure 1.

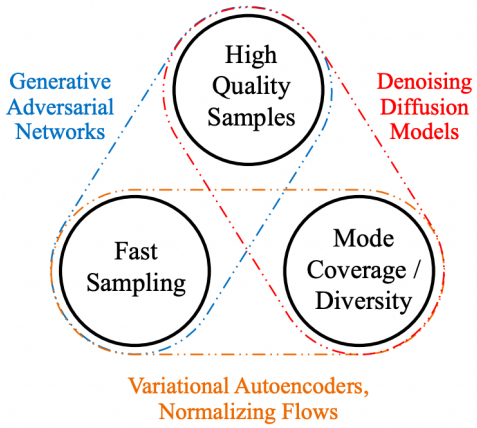


Figure 1: Taken from [7, p. 1]. Generative AI trilemma

A particular instantiation of a GAN model - Diffusion-GAN - attempts to bridge this gap using techniques from both models to improve performance for image generation tasks.

Goals. The original aim was to develop both a diffusion model (**DDPM**) and a GAN model (**StyleGAN2**) and compare them by using Fréchet Inception Distance (FID) and Inception Score (IS) metrics on three different datasets, as well as investing phenomena such as mode collapse. The extension was to develop Diffusion-GAN model as an improved GAN-based model, which uses the diffusion principle to improve performance. However, I have extended my goals to include:

1. More metrics such as improved precision and recall [8] and Kernel Inception Distance (KID), which will be described in section 3.5.
2. Dynamic tracking and analysis of the metrics during training.
3. Improving the DDPM model by adding more attention, in line with recent literature on the benefits of using attention [4], [9].
4. Adding improvements to Diffusion-GAN model by introducing a better embedding of the diffusion time step, aimed at improving the model's learning capability. (discussed later in Section 3.9.1).

2 Preparation

This section assumes some basic knowledge of deep learning, such as backpropagation and convolutions (see Appendix A for brief explanations).

2.1 Motivation for different Generative AI models

Let us assume that we want to create a generative AI producing high-quality images matching the distribution of images in some dataset $p_{data}(x)$. For example, let the dataset be CelebA - head shot images of celebrity faces [10]. For the model to generate images, it would need to estimate the dataset distribution by computing $p_\theta(x)$. A new image can be generated by sampling from $p_\theta(x)$. In theory it is possible to learn the probability distribution $p_{data}(x)$ directly through a neural network by the universal approximation theorem. The output layer would represent an image within the probability distribution of the dataset. This now imposes two restrictions on the output:

1. The output $p_\theta(x) \geq 0 \; \forall x$
2. It has to be normalized over the whole distribution to produce a valid probability distribution $p_\theta(x)$. The normalising constant would be: $\int_x p_\theta(x)dx$

The first condition can be easily achieved by applying arbitrary activation function $\sigma(z) > 0 \quad \forall z \in \mathbb{R}$. However, the second condition proves to be more challenging to satisfy. To achieve the second condition the output must be normalized so it has to be divided by a constant $\int_x p_\theta(x)dx$. As it is an integral over extremely high-dimensional pixel space, direct Monte Carlo computation is intractable. Attempts to overcome this problem have given rise to a variety of different approaches to generative modeling as a way to approximate or recreate the original distribution.

Explicit-density models. These define a tractable density $p_\theta(x)$ and optimize it directly via maximum likelihood or indirectly through Evidence Lower Bound (ELBO - see section 2.2.2). DDPM uses the latter approach - maximising the lower bound of the likelihood.

Implicit-density model. Instead of computing $\log p_\theta(x)$ directly, they avoid explicit normalisation by learning to sample from $p_\theta(x)$. Examples include GANs, which use an adversarial game to match distributions [11]. However, the lack of an explicit notion of coverage of p_{data} can lead to issues like mode collapse [12], as will be discussed in section 2.3.

2.2 Diffusion model

Diffusion models are a class of a generative AI algorithms that create data such as images by reversing a noising process. For image generation tasks, a small amount of noise is added over many steps and then a network is trained to reverse that process. There are three main components in the diffusion model: **the forward process**, **the reverse process** and **the sampling process**. The diffusion model works by sequentially introducing noise to the original data, slowly corrupting and destroying any structure in it, this process is called forward diffusion [13]. This noise is usually taken from a Gaussian normal distribution. Forward diffusion is applied to each sample until it is reduced to pure noise. Our goal in the diffusion model is then to train a network, which can help us reverse the forward diffusion process and iteratively remove noise. The idea behind this is that after a model is trained to denoise it, random noise sampled from a normal distribution can be passed to the model in an attempt to denoise it. It should produce some meaningful sample that has structure similar to the images in the original dataset. This is feasible as there is a well-defined notion of a "good" image at each denoising step and during training the model iteratively approaches the true distribution of the dataset.

Scheduling. This refers to how much noise is applied at each iteration of the forward process to the sample. It scales the mean and the variance of the Gaussian used to sample and add noise at each timestep in the forward process. Whilst the original DDPM model implemented linear scheduling, it was shown by OpenAI researchers [14] that, in the vast majority of cases, linear noise scheduling did not produce optimal results and yielded slower training when compared to other alternatives [14]. Linear scheduling was not optimal as it wasted the model learning capacity by introducing too much noise at early timesteps [15]. This results in later timesteps having very little meaningful structure that can be learned by the model. Instead, researchers at OpenAI used other schedulers, such as cosine, which achieved better results. In this project, however, I chose to stick to linear scheduling [13] to be consistent with the DDPM implementation.

Architecture. For the neural network to denoise the image and estimate the reverse process, the original DDPM implementation used UNet architecture [13], as UNet at the time achieved SOTA results for segmentation and denoising tasks in medical imaging [16]. UNet uses encoder - decoder structure where it compresses the input into a latent space and reconstructs output that is of the same size and dimensionality as the input data.

2.2.1 Forward process

The image is defined as $\mathbf{x}$ and the subscript determines the time step in the diffusion process. x_0 is the original image at $t = 0$ without any noise injected and x_T is the image reduced to pure noise sampled from Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The forward process is defined as $q(x_t|x_{t-1})$, it takes the image at $t-1$ and applies incremental Gaussian noise to produce an image at t. Similarly, the reverse process is defined as $p(x_{t-1}|x_t)$, which takes the image at t and produces an image at $t-1$. The forward and backward processes are shown in Figure 2 below.

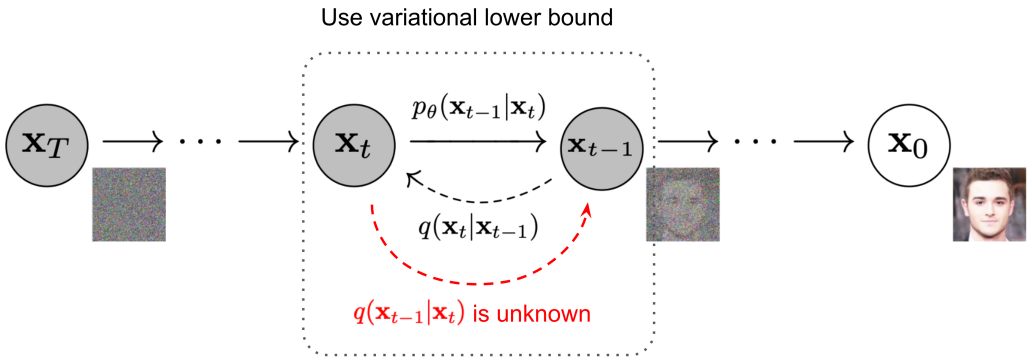


Figure 2: Taken from [17]

The forward process can further be expanded as such: $q(x_t|x_{t-1}) = \mathcal{N}(x_t, \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$. The benefit of rewriting the process in this form is that we can spot that instead of iteratively applying Gaussian noise to get from x_0 to x_t we can use a reparametrisation trick to arrive at a more general expression:

$$\mathcal{N}(x_t, \sqrt{1-\beta_t}x_{t-1}, \beta_t I) = \tag{1}$$

$$\sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon = \quad (\epsilon \text{ sampled from } \mathcal{N}(0,1)) \tag{2}$$

$$\sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon = \quad \alpha_t = 1-\beta_t \tag{3}$$

$$\sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon = \tag{4}$$

$$\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \quad \bar{\alpha}_t = \Pi_{s=1}^{t}\alpha_s \tag{5}$$

Note that the final equation allows us to generate $x_t$directly without having to iteratively compute previous x_{t-1}, x_{t-2}.... This removes a lot of redundant sequential computation and thus improves the performance of the model.

2.2.2 Reverse process

The goal of the diffusion model is to attempt to predict the reverse process.

$$p(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{6}$$

μ_θ and Σ_θ would be neural networks attempting to predict mean and variance. However, it was shown that variance can be derived effectively through fixed scheduler and predicting variance does not justify the minor performance increase [13]. For the loss function, since calculating the true posterior distribution and maximising the $\log p(\mathbf{x}_0)$ would be intractable, instead we can introduce Evidence Lower Bound (ELBO), which provides a direct lower bound on $p(\mathbf{x_0})$.

Definition

The **Evidence Lower Bound (ELBO)** provides a lower bound on the likelihood of distribution $p(X)$. Maximising ELBO would indirectly maximise the posterior, by maximising its lower bound.

Putting this definition into formal notation and applying it to the distribution over T timesteps:

$$L_{\text{ELBO}} = \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \leq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log p(\mathbf{x}_0)\right] \tag{7}$$

This can be further decomposed and rewritten as,

$$L_{\text{ELBO}} = \mathbb{E}_q\Bigg[\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} + \sum_{t>1}\underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} + \underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\|p(\mathbf{x}_T))}_{L_T}\Bigg]$$

Where L_{t-1} term trains the network to perform a single reverse process step. As shown in [13, p. 3] after a specific parametrization by the analysis of L_t.

$$= \mathbb{E}_{x_0,\epsilon}\left[\frac{1}{2\sigma_t^2}\left|\left|\frac{1}{\sqrt{\alpha_t}}(x_t(x_0,\epsilon) - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon) - \mu_\theta(x_t(x_0,\epsilon),t)\right|\right|^2\right]$$

This can further be simplified by parameterizing $\mu_\theta(x_t,t)$ to obtain a simplified expression and further simplifying, where t is uniform on the interval $[1,T]$, shown by [13].

$$L_{simple}(\theta) := \mathbb{E}_{t,x_0,\epsilon}\left[||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2\right] \tag{8}$$

Below is an outline of steps during training of the model and sampling to obtain an image from random noise. Sampling iteratively applies the reverse process to obtain an image x_0 that is $x_T \to x_{T-1} \to \cdots \to x_0$, where $x_T \sim \mathcal{N}(0,\mathbf{I})$.

Algorithm 1 Training

1: Initialize neural network ϵ_θ with parameters θ
2: **repeat**
3: Sample $x_0 \sim q(x_0)$ from the dataset
4: Sample $t \sim \text{Uniform}\{1,\ldots,T\}$
5: Sample $\epsilon \sim \mathcal{N}(0,I)$
6: Compute $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$
7: Compute gradient step on $\nabla_\theta\|\epsilon - \epsilon_\theta(x_t,t)\|^2$
8: **until** converged

Algorithm 2 Sampling

1: Sample $x_T \sim \mathcal{N}(0,I)$
2: **for** t = T, ..., 1 **do**
3: Sample $z \sim \mathcal{N}(0,I)$ if $t > 1$, else $z = 0$
4: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)\right) + \sigma_t z$
5: **end for**
6: **return** x_0

2.2.3 Neural network

The network, conditioned on a time step t, takes in an input noised image x_t and returns the predicted noise, which has the same dimensions and size. As mentioned previously, UNet [16] is a great candidate for this task to predict ϵ_θ. It uses UNet blocks as well as residual connection for better gradient flow [18] and to promote the learning at different feature levels. As it is dependent on t, it needs to be passed in as a parameter. However, passing in raw or normalized $[0, 1]$ scalar t would be detrimental: it would be difficult for the network to model the non-linear, long range dependency on t and the impact on all pixel values. It also does not allow for extrapolation after being normalized as values would have to be re-normalized again, invalidating previous data. Lastly, it would also result in poor gradient flow. The method for solving this issue is for the model to learn the embedding by first obtaining Sinusoidal Positional Embedding and then designing a simple MLP to project it to a higher-dimensional, differentiable representation.

Definition

Sinusoidal Positional Embedding is a technique first introduced in Transformer models [9, p. 6], used to map discrete values to a continuous vector of dimension d using different frequencies of sine and cosine functions. In the case of DDPM it is used to encode the time step t. The formula for computing is:
For $i = 0, \ldots, \frac{d}{2} - 1$

$$TE_{pos,2i} = \sin(\frac{t}{10000^{2i/d}}), \quad TE_{pos,2i+1} \cos(\frac{t}{10000^{2i/d}})$$

2.3 GAN model

2.3.1 Classic GAN

First introduced in 2014 [11], GANs are known for their ability to generate high-quality synthetic data including images, videos, and data augmentation. GANs are also well suited for image augmentation using techniques such as style transfer [19], used in StyleGAN [20] and StyleGAN2 [21] models, the latter of which will be implemented.

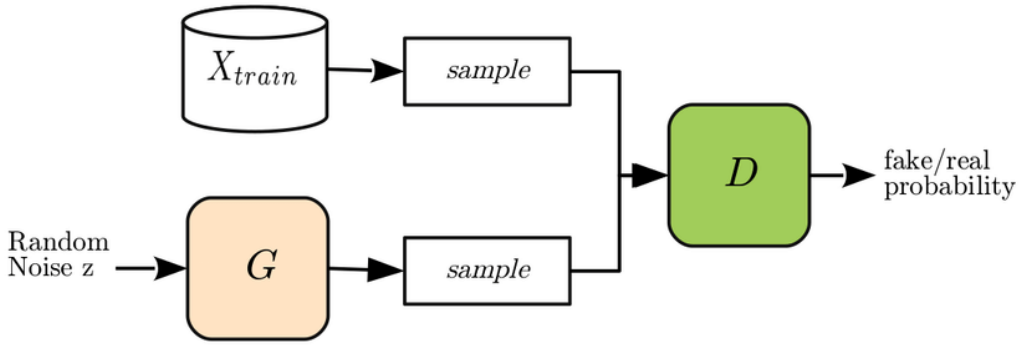


Figure 3: Taken from [22, p. 2]. Random noise z is sampled from $\mathcal{N}(0, \mathbf{I})$. z is then mapped, using generator $\mathbf{G}$, to a sample in image space with the same dimensions as images in the training dataset X_{train}. Discriminator $\mathbf{D}$ is exposed to a minibatch of real and generated samples. The loss is then computed according to the loss function described below.

All GAN models use a generator $\mathbf{G}$ and a discriminator $\mathbf{D}$. The two networks are trained in tandem. The discriminator is trained to classify images as real or generated, while the generator is trained to fool the discriminator. The adversarial nature of the technique results in the set-up modeled as a minimax game with the following value function V. [11]

$$\min_{G} \max_{D} V(\mathbf{D}, \mathbf{G}) = \mathbb{E}_{x \sim p_{data}(x)}[\log(\mathbf{D}(x))] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - \mathbf{D}(\mathbf{G}(z)))] \tag{9}$$

Where x is the true data and z is noise that is used as input for the generator. Ideally, this should achieve equilibrium where $D(x) = 0.5$ for both real and generated images, meaning that the generated images match the real image distribution and the discriminator is left to guess 50/50. This is shown further in figure 4, along with mapping from z to image space.

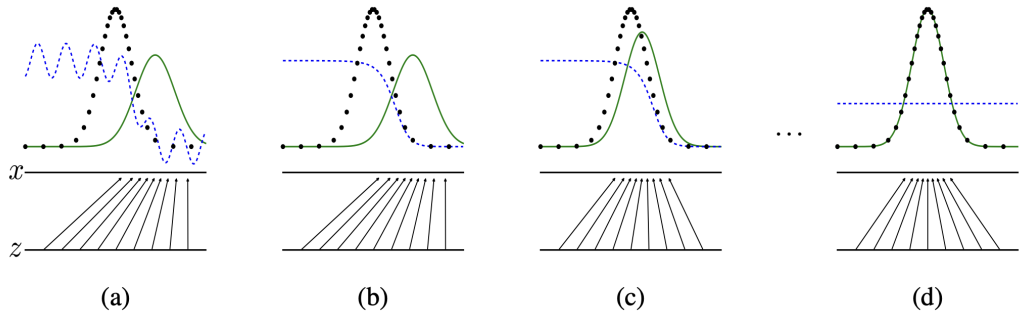


Figure 4: Taken from [11, p. 3]. As the training continues, the generative distribution $p_g(G)$(green, solid line) approaches the data generating distribution p_x(dotted black line), and when they are identical $p_g = p_{data}$ the discriminator is unable to differentiate, i.e $D(x) = \frac{1}{2}$(blue dashed line). The lower horizontal line z is the domain from which z is sampled, which is the random Gaussian noise input into the generator $G(z)$ and the upper horizontal line is the domain of x. The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_g on transformed samples.

It is also required to show that $p_g = p_{data}$ is the global minimum of the value function.

Recall the value function $V(G, D)$. First, consider a fixed generator G.

$$V(G, D) = \int_x p_{data}(x) \log(D(x))dx + \int_z p_z(z) \log(1 - D(G(z)))dz \tag{10}$$

$$= \int_x p_{data}(x) \log(D(x)) + p_g(x) \log(1 - D(x))dx \tag{11}$$

It was shown in [11] that a function $y \to a \log(y) + b \log(1-y)$ achieves maximum on $[0, 1]$ at $\frac{a}{a+b}$, thus through substitution we get $\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$. Using this result the training criterion for G becomes:

$$\min_G V = \mathbb{E}_{x \sim p_{data}} \log \left(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \right) + \mathbb{E}_{x \sim p_g} \log \left(\frac{p_g(x)}{p_{data}(x) + p_g(x)} \right) \tag{12}$$

Global Optimality 1 (Goodfellow, [11])**.** *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value* $-\log(4)$

Thus, from the result 1, the probability distribution of the generator data p_g will approach the real distribution p_{data} as the training converges, eventually reaching the Nash equilibrium of the minimax game.

A major concern with GAN models is the optimal discriminator problem [23]. If the discriminator learns significantly faster than the generator and gains high confidence, that is, $D(x) \to 1$ and $D(G(z)) \to 0$, then $\nabla_G \log(1 - D(G(z))) \approx 0$ - causing instability during training and the vanishing gradient problem. This is something that a lot of recent literature on GAN attempts to tackle, we will see some of the ways that more recent models such as Diffusion-GAN [5] and StyleGAN2 [21] deal with this in later chapters.

Definition

The **Vanishing gradients problem** arises in deep neural networks, when gradients become extremely small during backpropagation, resulting in particularly small updates to weights in deeper layers. This slows down the training or completely stops it.

Training. The training algorithm for GAN is shown below. In contrast to DDPM, it does not need to iteratively sample and requires only a single pass of the network to produce an image.

Algorithm 3 Minibatch stochastic gradient descent training of generative adversarial nets

Require: The number of steps to apply to the discriminator k, minibatch size m

1: **for** number of training iterations **do**

2: **for** k steps **do**

3: Sample $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\} \sim p_g(\mathbf{z})$ a batch of prior samples

4: Sample $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\} \sim p_{\text{data}}(\mathbf{x})$ a batch of real data samples

5: Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(\mathbf{x}^{(i)}) + \log \left(1 - D(G(\mathbf{z}^{(i)})) \right) \right]$$

6: **end for**

7: Sample $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\} \sim p_g(\mathbf{z})$ a batch of prior samples

8: Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D(G(\mathbf{z}^{(i)})) \right)$$

9: **end for**

Sampling. GAN sampling involves randomly sampling $\mathbf{z} \sim \mathcal{N}(0, I)$ and then sampling $\sim p_g(\mathbf{z})$ directly to obtain an image. Unlike in DPPMs, sampling occurs only once. A commonly used technique is the **truncation trick** [12], which truncates $\mathbf{z}$ to be within a certain range of the origin. This sacrifices image diversity for quality.

Mode collapse. This refers to a phenomenon in which the model fails to capture all modes of the data, hence the name. GANs attempt to approximate the mapping from the latent space to the true distribution space $G : \mathcal{Z} \to \mathcal{X}$. However, it does not necessarily recreate the entire probability density of $\mathcal{X}$. The generator is optimized to maximize the discriminator error. As mentioned in Section 2.1, classic GAN does not have the notion of coverage. This can result in a condition called mode collapse when the generator maps multiple $z \in \mathcal{Z}$ to a single $x \in \mathcal{X}$. The implication of that for image generation tasks is that the model produces realistic images of one type, yet does not capture the vast variety of the rest. We will see in Section 2.3.2 how that is mitigated in StyleGANs and further mitigated in Diffusion-GAN.

2.3.2 StyleGAN

StyleGAN incorporates novel techniques and improvements to the classic GAN. It builds on previous work that was released as an improvement to GAN [11], specifically to alleviate its issues concerning very unstable training and sensitivity to hyperparameters. For example, DCGAN [24] introduced deep convolutional layers and architectural guidelines for stable training, whilst ProGAN contributed progressive resolution growth [25] (see also WGAN [26]). In addition to adding further orthogonal improvements [24], [25], StyleGAN introduced the concept of **style**-modulated generation process, where an intermediate latent space $\mathcal{W}$ was introduced to represent styles or features, that influence image generation.

This provided fine-grained control over image features such as color, pose, shape and texture. Additionally, it introduced a mapping network to obtain that said intermediate style representation and proposed using minibatch statistics such as standard deviation of the learned features to promote diversity in the generated samples [27]. A more detailed explanation and implementation of the changes introduced in StyleGAN models is provided in Section 3.7.

2.4 Requirements analysis

The following are the requirements in line with the original proposal of the project:

Core. Implement DDPM and StyleGAN2 models and train them on CelebA, STL10 and FashionMNIST datasets. Create a pipeline for evaluating FID and IS scores and perform quantitative comparison of the two models.

Extension. Develop a diffusion-injected GAN model as an improvement to StyleGAN2. Extending the base models.

As I completed the requirements and the project evolved, I was able to add modifications to the DDPM model and the Diffusion-GAN model.

2.5 Development methodology

For this project **iterative development** aligned well with the proposal. The iterative model worked well with evolving requirements and evaluation metrics, and exploratory generative model experiments. The work was split into two-week work packages (**WP**s), where each WP focused on a different overarching task. Each WP included five stages:

Planning. Select the model to work on and do model specific literature review.

Implementation. Code the model and implement the training loop.

Train/evaluate. Train the model on the cloud GPU. Download the weights and results.

Review. Analyse the results, note any issues or findings.

Adapt. Introduce final changes if needed and use the findings to plan ahead for future WPs.

The most important incorporated practices are listed below. They were essential to ensure smooth completion of the requirements and to prevent avoidable delays in work.

Version control and backup. To ensure that the project is well structured and reflects the iterative approach, I used `Git` and `Github`. `Overleaf` was used to produce the dissertation report, which was also backed up online.

Tests. Before the image quality metrics were implemented, tracking the loss during the training stage provided a way to numerically test the models. At the end of the training the images produced by each model were also visually inspected.

Hardware. For the hardware to train the models on I decided to go with *Lambda Labs*, instead of the HPC provided by the university, due to its long queue times and unnecessary level of added complexity [28].

Controlled experimental design. During all of the experiments the models were compared on the same datasets, scaled to the same resolution. Identical hardware and image processing pipelines used.

Licensing. I comply with the **NVIDIA Source Code License-NC** for StyleGAN2 and Diffusion-StyleGAN2, and the **MIT License** for Denoising Diffusion Probabilistic Models (DDPM), conducting all work solely for internal, non-commercial research without bundling the license text, as permitted by the license.

2.6 Language and libraries

For this project, I have decided to use `PyTorch` [29] as it provides a flexible framework for deep learning and is widely adopted within the research community and is very well documented. `Torchvision` was also used to streamline data preprocessing and data access. `Numpy` provided easy array operations and tensor-like operations within metrics and networks. For code formatting I used `yapf` to automatically format code to PEP 8 standard and improve code readability.

3 Implementation

3.1 Datasets

For the experiments, the following datasets were used:

- CelebA - contains 202,599 face images of various celebrities. Each image is 178×218 pixels. [10]
- FashionMNIST - consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Classes - T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot. [30]
- STL10 - 10 classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck. Images are 96x96 pixels, color. 500 training images (10 pre-defined folds), 800 test images per class. 100000 unlabeled images for unsupervised learning. [31]

The CelebA dataset was chosen for evaluating unconditional image generation in a relatively controlled setting. It consists only of celebrity face images that are well-aligned and centered, simplifying the generative task. The STL-10 dataset was chosen to assess model performance when faced with a complex, multiclass task. The Fashion-MNIST dataset, while also multiclass, is considerably simpler, with the image classes having less variation. Model performance on this dataset was used to exemplify domain mismatch.

3.2 Preprocessing

Torchvision library provides an interface to download and access smaller datasets such as STL10 and FashionMNIST. All images had to be mapped from $[0, 255]$ to a $[-1, 1]$ tensor of size `(B,C,H,W)`, where `B` - batch size, `C` - number of color channels(1 for grayscale and

3 for RGB), `H` - height, `W` - width. Images from the CelebA dataset were also cropped to maintain face alignment before being resized to 64x64. Resizing and cropping was done using torchvision transformations. However, using torchvision resizing could negatively impact training for feature extraction and result in less accurate FID scores due to aliasing. This is discussed in the evaluation section 4. Extra care was taken with FashionMNIST dataset as it is grayscale. Since the models developed in this dissertation were designed for 3 channel RGB images the FashionMNIST dataset had to be transformed $(\mathtt{B}, \mathtt{1}, \mathtt{H}, \mathtt{W}) \rightarrow (\mathtt{B}, \mathtt{3}, \mathtt{H}, \mathtt{W})$ by duplicating the grayscale value across other channels.

3.3 Environment and reproducibility

To ensure a fair comparison and judgment of the models, I decided to train them on the same isolated cloud hardware. All models were trained on Nvidia's **GH200** superchip with 64vCPUs and 432 GiB RAM. This allowed for a significant speed up in training and inference for all models. The models were trained for 30 epochs on each dataset, due to time and financial constraints. Typically, for a dataset such as CelebA to obtain SOTA results, it requires at least 500k training steps, which is significantly higher than the number used (see figure 5) [21] [32]. However, there are diminishing returns in training the models for longer and producing high-end fidelity images is not the goal of this dissertation. While the arbitrary number of 30 epochs may not provide the SOTA results, they should produce enough data points to analyse trends and convergence of each model.

Dataset	Number of samples	Number of training steps at 30 epochs
CelebA	202,599	189,930
STL10	100,000	93,750
FashionMnist	70,000	65,610

Figure 5: Number of images and training steps per dataset over the whole training (30 epochs)

3.4 Codebase

This section provides an overview of the structure and functionality of the codebase. This overview omits empty `__init__.py` files.

File / Folder	Description	Lines of code
train.py	Entry point for training, handles parsing CLI.	29
interpolation.py	Implements latent space interpolation	84
evaluate.py	Handles calculating the metrics post-training.	39
config_loader.py	Auxiliary module used to load configs	11
dataset.py	Provides an interface for loading the datasets.	66
models	Contains StyleGAN2, DDPM and Diffusion-GAN models.	2018
trainers	Contains trainers for the respective models. Handles training, metric tracking and saving logic.	669
configs	Contains YAML configuration files organized per model and per dataset, with one config for each model-dataset pair (9 configs)	135

Table 1: General overview of the projects folder.

The `models` folder contains 3 subfolders: `stylegan2, ddpm, diff_gan`, each following the structure shown in Table 2.

File / Folder	Description	Lines of code
diffgan.py	This files defines the entire model.	172
diffusion.py	Used for generating timestep distribution and applying it to images.	143
discriminator.py	Discriminator network architecture.	165
ema.py	Implementation of the Exponential Moving Average copy of the generator.	19
generator.py	Generator network architecture.	170
losses.py	Implements the discriminator and generator loss from the minimax objective.	13
mappingmlp.py	Mapping network architecture. Mapping $\mathcal{Z} \to \mathcal{W}$.	22
util.py	Auxiliary file containing Path Length Penalty and R1 implementations and commonly used blocks.	136

Table 2: Example of a model subfolder. `/models/diff_gan/` showcased above.

The `trainers` folder contains all of the training logic, metrics tracking and saving operations (see Table 3).

File / Folder	Description	Lines of code
calc_metrics.py	Implements tracking of training precision and recall by computing it from saved images.	209
ddpm_trainer.py	Contains DDPM training routine and metrics tracking.	84
diffgan_trainer.py	Contains Diffusion-GAN training routine and metrics tracking.	102
stylegan2_trainer.py	Contains StyleGAN2 training routine and metrics tracking.	96
util.py	Implements pipeline for loading and storing images and computing FID and KID on the fly.	74
metric_plots.py	Contains auxiliary plots used to create figures for this project.	89

Table 3: `trainers` subfolder.

3.5 Metrics

3.5.1 Choice of metrics

It is essential to have a benchmark to allow to compare different models and to argue relative improvements.

The most widely used metrics in image generation are the Fréchet Inception Distance (FID) and Inception Score (IS), both of which were originally designed to correlate with human evaluation [33]. Originally, I proposed FID and IS metrics to track model performance. Those metrics make it possible to evaluate the quality of images produced by generative models without human intervention as they use a large-scale InceptionV3 model. However, they have a bias towards different models, as different sampling algorithms produce different p_θ distribution at the same finite number of samples [34]. Inception Score also only makes sense to use for multi-class datasets as it relies on high confidence and diversity between classes. Another pitfall to watch out for is the domain mismatch - InceptionV3 was trained as a classifier on ImageNet [35], not specifically trained on human faces or different types of clothing. As a result, IS was only suitable for assessing performance on the STL10 dataset due to its multi-class nature and containing image classes that aligned with the domain of ImageNet.

I decided to implement additional metrics, such as Kernel Inception Distance, which is as an unbiased estimator, and is more stable at smaller sample sizes compared to FID [33]. I also decided to provide a confidence interval to the computation, as well as precision and recall [34] [36] to counteract some of FID's shortfalls [33]. I used Torchmetrics as a high-level API, which provides an wrapper for FID and KID metrics calculation. To access the InceptionV3, I installed torch-fidelity, which provides an interface to the weights and features of the model [37] [38].

3.5.2 Inception Score

Inception Score (IS), as first introduced in [27], is used to assess the performance of a generative model. It is calculated by passing the output of the generator through the Inception Network classifier, that was trained on an extremely large number of datapoints. It was also shown to correlate with human evaluation of picture quality [27]. The two proposed qualities that realistic images have are stated below:

1. Generator output should be easily classifiable, which means images will have low entropy of predicted class distribution.

2. Overall class distribution should have high entropy, this condition promotes diversity in the generated images.

These conditions led to the IS metric calculation. However, it fails to capture mode collapse, as high diversity of the classes in the generated images does not necessarily imply that the generator captures the full diversity of the underlying image distribution. The score is defined by:

$$\mathbf{IS}(G) = \exp(\frac{1}{N}\sum_{i=1}^{N} D_{KL}(p(y|x^{(i)}||\hat{p}(y))) \tag{13}$$

Torchmetrics provides an interface wrapper for calculating FID scores based on the number of features. To access the InceptionV3, I have installed torch-fidelity [38], which contains the download for the weights of the InceptionV3 used to calculate IS, FID and KID.

3.5.3 Fréchet Inception Distance

Fréchet Inception Distance (FID) was introduced as an improvement to the standard Inception score, that can be applied to a larger variety of datasets. It successfully tackled the issues of susceptibility to mode collapse by comparing the distribution of the generated image features to the distribution of original image features. FID also relies on the InceptionV3 model, but instead of focusing on final classifications of images it compares the learned features of the dataset. FID is an extension of the idea that the objective of generative learning is to obtain a model that produces data with a similar distribution of features to the original dataset [39]. A subtle but key difference between the two metrics is that FID has a larger domain than IS. The domain of IS is the 1000 ImageNET classes, while FID uses learned features, which can be interpreted as shapes, textures and colors. Using intermediate features provides a much bigger domain as it covers all datasets with natural shapes and textures. After measuring the distribution of features in two different datasets (real and generated), we can measure the Euclidean distance between the means and compute the similarity between the covariance matrices. FID is regarded as the more robust metric, as it captures the diversity of the generated data, allowing it to capture mode collapse [39].

$$\mathbf{FID} = \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r\Sigma_g\right)^{1/2}\right) \tag{14}$$

Figure 6: Distance between two multivariate Gaussian distribution (μ_r, Σ_r) and (μ_g, Σ_g), used in calculating FID.

The results are usually denoted as FID_x, where x is the number of samples in each dataset used to compute it. It is also important to keep in mind that FID and IS are inherently biased, when computed from a finite number of samples. In the case of FID, two different generative models can have the same p_θ at FID_∞. However, at a finite number of samples, one of them may produce a more fat-tailed distribution, have different variance, or exhibit non-Gaussian behavior impacting the score. This bias is persistent and each model can have different bias term. For this reason, it is important to pick the number of samples large enough to reduce the magnitude of the bias. [34] In the majority of the recent literature on generative AI, FID scores are referred to as FID_{50k}. Conditioning models on FID metric or optimizing them based on FID scores frequently results in overfitting as seen in Figure 7.

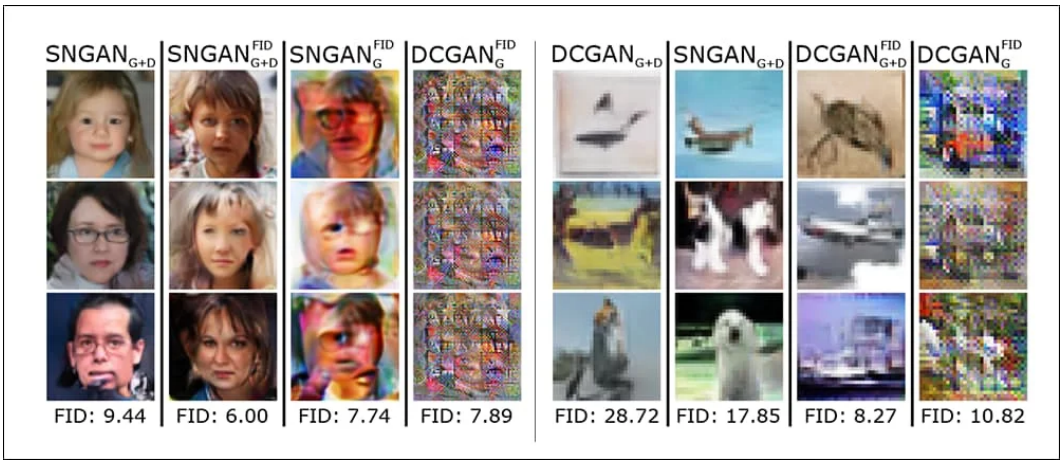


Figure 7: Taken from [40]. It shows that lower FID score $\neq$ better image quality. Models that used FID as part of its loss function produced significantly worse results despite obtaining lower FID score.

Although the majority of the literature still reports FID as the dominant metric for image quality, it has three disadvantages, as previously discussed. These are re-stated below:

Problem 1: It combines both image quality and diversity into one. Although this may be useful in some cases, it does not allow to isolate diversity and image quality.

Problem 2: Models that used FID during training tend to overfit to the metric.

Problem 3: It is a biased estimator making Gaussian assumptions [40].

Clearly, it is insufficient to compare models only using this metric given the wide range of different limitations exhibited by different models.

3.5.4 Kernel Inception Distance

KID is based on a similar idea to FID; it measures the distance between two different distributions - the distribution of extracted features by InceptionV3 in the true dataset and the distribution in the generated dataset. Unlike FID, however, KID does not make any Gaussian assumptions and instead uses Maximum Mean Discrepancy (MMD) [3] [36]. It is

[3] See appendix B on MMD

also common practice with KID to use randomly sampled subsets of images to obtain the standard deviation, which provides a confidence interval on the score. KID is also shown to be more numerically stable and accurate with smaller sample sizes [41], [36]. This was advantageous for my experiment since the DDPM inference time is slow. This allowed me to measure the true performance of DDPM with reasonable precision during training while being computationally feasible.

3.5.5 Precision & Recall

Precision and Recall scores, introduced in [8], aim to tackle the issues with FID and KID metrics caused by combining both diversity and realism of generated images. This means that a low FID or KID score can indicate high quality, high diversity, or anything in between. The proposed metric measures *precision*, which is the fraction of images that are realistic, and *recall*, which is the fraction of the training manifold covered by the generated samples.

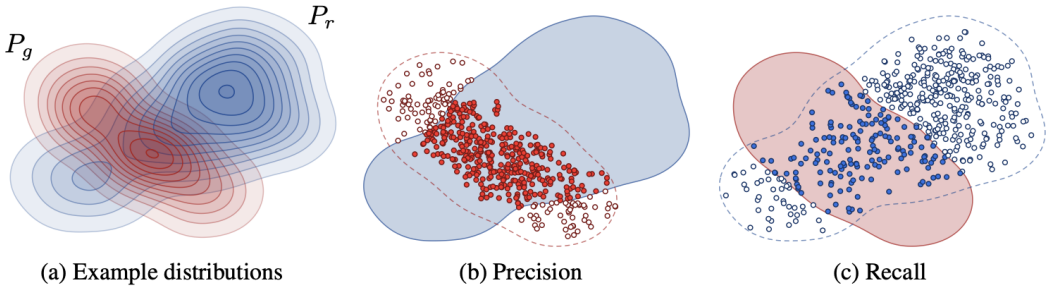


Figure 8: Taken from [8, p. 2]. (a) Denote the distribution of real images P_r (blue) and the distribution of generated images with P_g (red). (b) Precision is the probability that a random image from P_g falls within the support of P_r. (c) Recall is the probability that a random image from P_r falls within the support (see 3.5.5) of P_g.

Definition

The **support space** of a probability distribution is defined as the set of all possible values that a random variable can take having that probability distribution.

In practice, it is impossible to measure the exact support space of real or generated images. To avoid this limitation and approximate the support space to calculate approximate recall and precision, *k-nearest neighbors* is used. This allows one to measure how many generated images fall within the k-th nearest-neighbor manifold. This is a region defined around each $x \sim P_r$, that is, within a hypersphere that reaches the k-th nearest neighbor of x.

Now, to return the problems presented with the FID metric earlier, we can see that using Recall and Precision as two separate metrics provides much more insight into the models' performance. Precision captures how realistic the produced image looks [8], solving problem **1** of FID. Recall shows how much of the original data distribution is covered by the generated images. This allows us to capture mode collapse and overfitting, which is the problem **2** of FID. Finally, no Gaussian assumptions are made, which allows one to more accurately empirically model the real world, eliminating problem **3**.

To measure precision and recall for the models developed here, I saved 500 samples generated from each epoch of training. To allow for better accuracy while keeping it computationally feasible, 500 images were used from the real image distribution to measure the two quantities.

3.5.6 Latent space interpolation

Unlike the previously discussed quantitative metrics, latent space interpolation is not a numerical measure of performance but rather a qualitative procedure used to inspect and compare the structure of the latent spaces learned by generative models. By generating images from intermediate representations between two different latent vectors w_1 and w_2. This was done via linear interpolation in the latent space. Extra care had to be taken with picking w_1 and w_2 points, as the latent space was not guaranteed to be well behaved. This meant that after picking some initial w_1, using a random point small distance away from w_1, s.t $w_2 = w_1 + \epsilon$ may produce incoherent images. Randomly sampling w_1 and w_2 did not produce coherent results, as there is no guarantee about the distance between them being large and 7 steps will not be able to capture the gradual change. In the end, interpolation was implemented by picking an initial w_1 and repeatedly sampling w_2 until $||w_1 - w_2||_2 < \alpha$, where α was an arbitrary small threshold variable.

Algorithm 4 StyleGAN2 Latent space interpolation.
G is a the generator, **M** is the mapping network

Sample $z_1 \sim \mathcal{N}(0, I)$
$w_1 \leftarrow \text{MappingNetwork}(z_1)$
repeat
    Sample $z_2 \sim \mathcal{N}(0, I)$
    $w_2 \leftarrow \text{MappingNetwork}(z_2)$
until $||w_1 - w_2||_2 < \alpha$
for $\lambda \in \texttt{linspace}(w_1, w_2, steps)$ **do**
    $w_\lambda \leftarrow (1 - \lambda)w_1 + \lambda w_2$
    $Images[i] \leftarrow \text{Generator}(w_\lambda)$
end for

Algorithm 5 DDPM Latent space interpolation.

Sample $z_1 \sim \mathcal{N}(0, I)$
repeat
    Sample $z_2 \sim \mathcal{N}(0, I)$
until $||z_1 - z_2||_2 < \alpha$
$z_2 \leftarrow z_1 + \epsilon$
for $\lambda \in \texttt{linspace}(z_1, z_2, steps)$ **do**
    $z_\lambda \leftarrow (1 - \lambda)z_1 + \lambda z_2$
    $Images[i] \leftarrow \texttt{DDPM}_{\texttt{sample}}(z_\lambda)$
end for

3.6 Tracking training

A summary of all tracked metrics is provided in Table 4.

Metric	Purpose	Frequency
FID	Correlates to human evaluation across generated images	Every epoch
KID	Unbiased version of FID	Every epoch
IS	Measures how well the model captures multiclass distribution	Every epoch
Recall	Measures what percentages of the support space of the real distribution is covered	Every epoch
Precision	Measures how close the generated images are to real images	Every epoch
Log loss	Expected to reduce across training to indicate learning and show converging behavior	Every iteration
GPU memory alloc (MB)	Track GPU memory usage, helps to indicate leaks and computational requirement of each model	Every epoch

Table 4: Summary.

3.7 StyleGAN2

StyleGAN2 is based on top of previously discussed GAN [21], [42]. It implements the same adversarial game principle. However, it introduces several significant changes:

1. **The discriminator architecture is changed**: the discriminator became a deep convolutional neural network. Proposed injecting minibatch statistic such as minibatch standard deviation to prevent mode collapse.

2. **Introduced mapping network**: instead of feeding z directly to the generator, it is first passed through an MLP to obtain an intermediate representation w, which allows more control over styles.

3. **The generator architecture is changed**: the generator also became a deep convolutional neural network. *Modulation* and *demodulation* are introduced, which control the convolutions and allow for fine-grained control over features at different scales.

4. **Loss function is changed**: extra terms in the loss function such as *R1* and *Path Length Penalty* are introduced to promote better control over style and justify the introduction of intermediate representation w.

3.7.1 Architecture

Mapping Network. Mapping network can be represented as a function

$$f : \mathcal{Z} \to \mathcal{W}$$

where

- $\mathcal{Z} \subseteq \mathbb{R}^n$ is the input latent space, sampled from an isotropic Gaussian ($z \sim \mathcal{N}(0, \mathbf{I})$)
- $\mathcal{W} \subseteq \mathbb{R}^n$ is the disentangled latent space
- f is the MLP mapping network

The mapping network is an 8-layer MLP that uses LeakyReLU, instead of normal ReLU, to better handle negative inputs[4]. Equalized linear layers were also used to improve the weight initialization and to ensure that the variance of output across layers is roughly constant. The equalized linear layer causes the weight matrix to be scaled st $W_{equalized} = \frac{W}{\sqrt{fan-in}}$ where fan-in is the number of features in the previous layer. I opted to use the value $n = 512$, in line with Karras [21], where it was shown to be the optimal value when tested for similar datasets and resolution.

Generator. The generator architecture can be broken down into 4 main blocks:

Style block. One of the main building blocks of the generator. As shown in Figure 9 below, a style block is a specialised module used to inject style information. It takes in an intermediate vector w and passes it through a learned transform to obtain the style $s = A(w)$, which is then used to scale convolution weights through what is called *modulation*. It also typically normalises the weights by unit variance, which is called *demodulation*. Demodulation helps to prevent activations from blowing up and any one style becoming too dominant. On top of that, it also adds trainable noise and a bias term, followed by LeakyReLU, providing non-linearity and control over coarse and fine styles independently.

toRGB block. It contains only a single *modulated* 1x1 convolution for the purpose of converting features to an RGB image. The result of RGB blocks at different resolution levels is then summed to produce the final image.

Generator block. This is the biggest building block. It consists of 2 style blocks followed by an RGB block and an upsample layer[5] to increase the spatial resolution. In this way, each generator block operates on a different spatial resolution and feature space.

[4] See appendix A for explanation
[5] See apendix for more on upsample layer

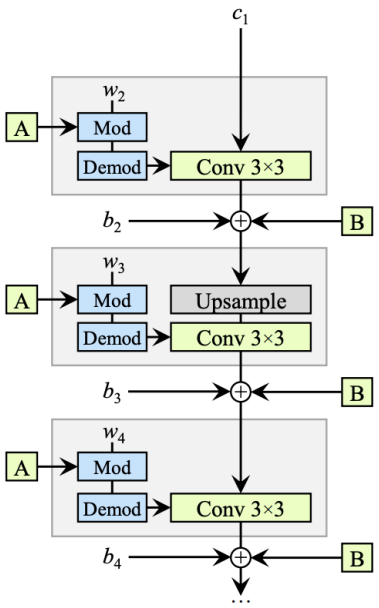


Figure 9: Taken from [21, p. 3]. Each of the gray boxes on the picture is a **style block** + optional upsample layers. **A** denotes the learned vector in disentangled space $\mathcal{W}$, which is produced by the mapping network. **B** is random noise injected across color channels. w, b, c are learned weights, biases and constant input respectively.

Putting it all together: Instead of starting with some initial noise like in the classic implementation of GANs, the generator starts with a learned $4x4$ constant. This produces a variety of images because the randomness comes from modulated convolutions using a learned-style vector $s = A(w)$. Then a single style block is applied before applying the $\log_2 x - 2$ generator blocks (one at each resolution). For example, for a 64x64 resolution dataset, the model would start with a 4x4 learned constant, single style block, and 4 generator blocks responsible for 8x8, 16x16, 32x32, 64x64 resolutions. An RGB block performs a 1x1 convolution to produce an image of the current resolution with 3 color channels. At the end the values from RGB blocks at each resolution are summed to produce the final image.

Discriminator. The discriminator structure resembles that of the generator, but in reverse. It can also be broken down into three main components:

Equalized convolution. This follows the exact same principle as equalized linear layers from the mapping network. Weights in the convolution kernel are scaled s.t $W_{equalized} = \frac{W}{\sqrt{fan-in}}$. Applying scaled weights combats instability from differing parameter scales across layers and ensuring all layers learn at similar rate [25].

fromRGB block. The 3 RGB channels are converted to feature channels using a 1x1 equalized convolution.

Discriminator block. Similarly to generator blocks, each discriminator block operates on a different spatial resolution. The block contains two equalized convolutions to extract features followed by a downsample layer[6] to reduce the resolution for the next block.

[6]See apendix for downsample layer explanation

The discriminator also uses residual connections, which adds the original input to the output of the block (see Section 3.8 for a more detailed description of residual blocks; see also Figure 10).

Putting it all together: The discriminator initially converts the three color channels of the picture to n feature channels using *fromRGB*. 64 feature channels chosen according to Karras [21], considering that the model is not trained on ultra-high resolution images and the number of parameters, this value allows to capture an appropriate range of features. It is then followed by $\log_2(res) - 1$ discriminator blocks. Residuals are used to prevent vanishing gradients and stabilize training. After applying the discriminator blocks, it then uses minibatch standard deviation, to calculate per pixel deviation to promote diversity, calculated using:

$$\mu_i = \frac{1}{N} \sum_g x_{g,i} \tag{15}$$

$$\sigma_i = \sqrt{\frac{1}{N} \sum_g (x_{g,i} - \mu_i)^2 + \epsilon} \tag{16}$$

Using minibatch standard deviation helps to improve mode coverage and reduce the likelihood of mode collapse [25]. Lastly, the discriminator applies the final convolution and returns the classification score.

3.7.2 Training

Algorithm. The main training loop follows the algorithm described previously (see Algorithm 3). There are two additions to the training algorithm.

- R_1 gradient penalty applied only to the real data in the discriminator to ensure local smoothness and continuity around real samples. [43]

  $$R_1 = \frac{\gamma}{2} \mathbb{E}_{x \sim p_{data}} ||\nabla D(x)||^2$$

  γ - scaling coefficient for the R_1 loss. The R1 loss. The gradient penalty is computationally expensive as it involves computing second derivatives, hence I only implemented every 32 iterations and it is further scaled by the skipped iterations. On a more intuitive level, this penalizes large changes in discriminator confidence for small changes in real images. This in turn promotes smoother decision boundary and prevents the discriminator from overfitting.

- Path length penalty is used to regularize the generator. The idea is to encourage that a fixed-size step in the latent space $\mathcal{W}$ results in a non-zero, fixed-magnitude change in the image.

  $$L_{path} = \mathbb{E}_{w,y \sim \mathcal{N}(0,I)} (||\nabla_w G(w) y||_2 - a)^2$$

  This helps "disentangle" latent space $\mathcal{W}$, which produces a more consistent style control.

Stabilizing training. The Exponential Moving Average (EMA) copy of the generator weights [44] was used to stabilize training and used for evaluation metrics such as FID and

KID according to $\theta'_{EMA} = \beta \cdot \theta_{EMA} + (1 - \beta)\theta_G$.

Hyperparameters. The majority of the hyperparameters were kept constant across the datasets. Mapping network was kept at 8 layers to produce 512 dimensional vector in $\mathcal{W}$ and $\mathcal{Z}$ is also 512. Style mixing probability was kept at 0.99. R_1 regularization occurred after every 32 minibatches with $\gamma = 10$. Path penalty was applied every 32 minibatches after 5000 minibatches. Path length regularization interval of 32 minitbatches. The generator and the discriminator learning rate was set to 0.001, while the mapping net was set to 0.005. All of the models were trained for 30 epochs on each dataset. The Exponential Moving Average copy of the model was kept with 0.99 decay constant.

3.8 DDPM

The overall structure of the DDPM model follows that outlined in Section 2.2. The model was further modified by adding more *attention*. This choice was motivated by recent literature showing improvement in image generation tasks by leveraging the attention mechanism [1], [15], [45], [46].

3.8.1 Architecture

DDPM consists of a UNet conditioned on a timestep t to predict noise ϵ_θ for the reverse process $p_\theta(x_{t-1}|x_t)$ [13] [16]. UNet can be broken down into three parts: an encoder, a bottleneck and a decoder. The most important reused component within the DDPM network is a ResNet block, shown below in figure 10 [18].

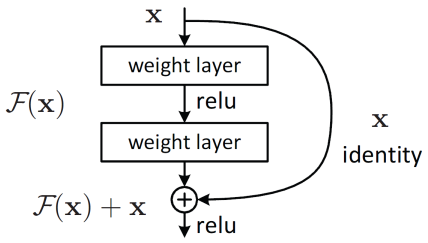


Figure 10: Taken from [18, p. 2]. ResNet block. This block is crucial to prevent vanishing gradients and enabling deeper models. ResNet is also more suited for capturing small perturbations in the data in line with the DDPM objective of predicting noise ϵ_θ.

In DDPM, the residual block contains two 3x3 convolutions, with group normalization (normalize activations across features to prevent instability in training) and SiLU activation[7] for smooth gradient propagation. DDPM also uses attention [9] to capture long-range spatial dependencies in the picture to ensure that it is encoded properly to produce a consistent image.

[7]See appendix for SiLU explanation

Definition

Attention is a machine learning technique that computes relative importance of each component in a sequence. In the case of image generation, it allows the model to produce consistent images by allowing any pixel or region to incorporate information from other distant parts of the image.

The structure of the encoder, decoder and the bottleneck are illustrated in Figure 11.

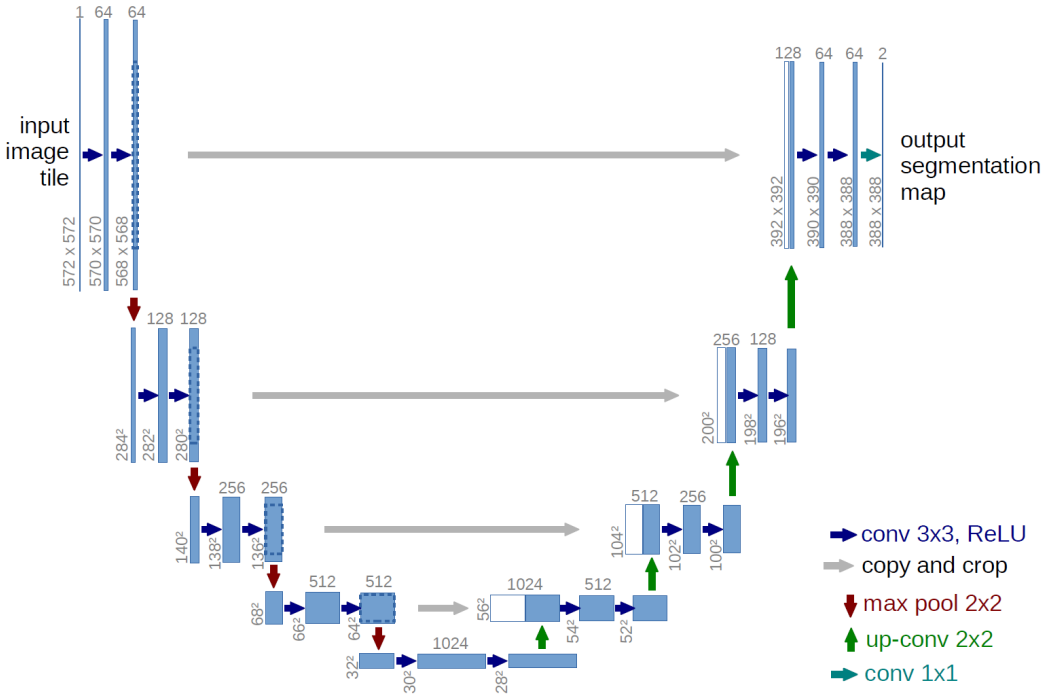


Figure 11: Figure taken from [16, p. 2]. This shape is where UNet gets its name from. On top of the feedforward connections there are residual connections (grey arrows), which helps preserve spatial and local information when decoding. This also helps to mitigate the vanishing gradient problem [18]. Max pool 2x2 is a specific implementation of the downsample layer, which applies a 2x2 kernel that computes the max pixel value in the region to downsize an image.

Encoder. The encoder progressively reduces the spatial resolution of the image, while simultaneously expanding its feature channels, producing a high-dimensional compact representation of the image. This contains a sequence of encoder blocks at specified resolutions. Each encoder block contains two residual blocks with time embedding followed by linear attention and a downsample layer, which is different to the original DDPM implementation [13], as I use more attention within the model and apply it at every resolution, rather than only at lower resolution blocks. This is in line with [1] introducing more forms of attention to diffusion models. More attention increases learning capacity but also increases the number of parameters and has $O(n^2 d)$ time complexity, where $n = H \times W$ and $d = C$ in (C, H, W). Each encoder block multiplies the number of features and decreases (H, W) dimensions of the image. For example, the first encoder block will transform the three color channels of an image to N feature

dimensions and each subsequent block will multiply that: $(3, H, W) \rightarrow (N, \frac{H}{2}, \frac{W}{2}) \rightarrow (2N, \frac{H}{4}, \frac{W}{4}) \rightarrow \dots$

Bottleneck. This is also known as the middle block and consists of a single residual block followed by attention and another residual block. The bottleneck processes lowest spatial resolution with highest number of channels. It uses self-attention to model long-range dependencies in the image to preserve structure. It is also computationally cheap due to it operating on lower spatial resolution.

Decoder. The decoder can be thought of as the inverse of the encoder. Each decoder block has 2 residual blocks, followed by linear attention and upsample layer to recreate the spatial resolution of the input image. The decoder can be thought of as the inverse of the encoder, so it will reduce the feature dimensions until the image space (3 color channels) and increase (H, W) to replicate the original dimensions. After each block $(C \cdot N, \frac{H}{2^C}, \frac{W}{2^C}) \rightarrow ((C-1)N, \frac{H}{2^{C-1}}, \frac{W}{2^{C-1}}) \rightarrow \dots \rightarrow (3, H, W)$

Finally, the tensor needs to be converted to an image of size (C, H, W), which is done by applying a 1x1 convolution to convert it to the desired resolution and 3 color channels. In the above explanation, I have left out how the network was conditioned on the timestep t. That was done by obtaining an appropriate representation or *embedding* for the time step. ResNet block then simply adds the value to the input of the block. I have implemented learned embedding, which uses a small fully connected network to map the scalar t to higher dimensional representation. However, this embedding style is now outdated, so I then reimplemented it with sinusoidal positional embedding, which produced better results [9].

3.8.2 Training

Algorithm Training follows the algorithm described earlier in Algorithm 1. Function `p_sample(...)` was written to do step-by-step DDPM sampling to calculate

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) \tag{17}$$

This was then used within a loop to start from pure noise $x_T \sim \mathcal{N}(0, \mathbf{I})$ and using $\texttt{p\_sample}(x_t)$ to calculate $x_{t-1}, x_{t-2} \dots, x_0$ and obtain an image resembling the original distribution.

Hyperparameters Similarly to StyleGAN2, most hyperparameters were kept constant across datasets. ADAM optimizer was used with 0.001 learning rate. Linear noise scheduling with start $\beta_{start} = 0.0001$ and maximum noise $\beta_{end} = 0.02$. The forward diffusion was set to 300 timesteps as it is a common value allowing for faster inference time than 1000 steps and still provides enough capacity for learning [13]. Initial feature dimension was set to 64 and scaled by $(1, 2, 4)$ channel multipliers in each respective block.

3.9 Diffusion-GAN

Diffusion-GAN is heavily based on the previously implemented StyleGAN2. It introduces new orthogonal improvements, aimed at further stabilising GAN training, which is known to be particularly unstable [26], [12], [11]. The improvements introduced are borrowed from

diffusion models, but are used in a different way. In this model, the forward diffusion is used as an image augmentation technique. It is used to control and synchronise the discriminator and generator learning speed to promote training stability. This type of augmentation provides a fine-grained level of control compared to other augmentations, such as image distortion, as the amount of noise is easily adjustable. A new metric is introduced to measure the discriminator overfitting, which is used directly to scale the amount of noise. This makes the job harder for the discriminator and discourages overfitting [5].

3.9.1 Architecture

The Diffusion-GAN variation implemented here was built on top of the StyleGAN2 model that I have already implemented. The idea behind Diffusion-GAN is to further stabilise the training and improve perceptual image quality through the introduction of noise injection similar to diffusion models. Noise is injected into both real and generated samples to prevent the discriminator from overfitting and preventing the optimal discriminator problem, which causes vanishing gradients for the generator. Diffusion-GAN uses the same initial Markov assumption about diffusion, and further models each sample y as having instance noise injected as a T-component Gaussian mixture distribution over all noise levels.

Definition

A **T-component Gaussian mixture distribution** is a probabilistic model defined as:

$$p(x) = \sum_{t=1}^{T} \pi_t \mathcal{N}(x \mid \mu_t, \Sigma_t)$$

where each π_t is a mixing coefficient such that $\sum_{t=1}^{T} \pi_t = 1$, and $\mathcal{N}(x \mid \mu_t, \Sigma_t)$ is a multivariate Gaussian distribution with mean μ_t and covariance Σ_t.

More specifically:

$$x \sim p(x), y \sim q(y \mid x) := \sum_{t=1}^{T} \pi_t q(y \mid x, t) \tag{18}$$

$$q(y \mid x, t) = \mathcal{N}(y; \sqrt{\bar{\alpha}_t} x, (1 - \bar{\alpha}_t)\mathbf{I}) \quad \text{from 1} \tag{19}$$

Naturally, as an improved version of StyleGAN2, it uses the same mini-max game mechanic, which yields a similar objective.

$$\begin{aligned} V(\mathbf{G}, \mathbf{D}) =& \mathbb{E}_{x \sim p(x), t \sim p_\pi, y \sim q(y|x,t)}[\log(\mathbf{D}(y, t))] + \\ & \mathbb{E}_{z \sim p(z), t \sim p_\pi, y_g \sim q(y|\mathbf{G}(z),t)}[1 - \log(\mathbf{D}(y_g, t))] + R_1 + L_{path} \end{aligned} \tag{20}$$

Although this modified objective is very similar to the original GAN formulation, the addition of perturbed y_g means we must now prove that minimizing it actually reduces the divergence between the real and generated images instead of the perturbed samples. Indeed, this was proven by Wang [5]. It is now important to grasp how diffusion is used to aid the learning process. Previous work outlined that a lot of the instability in GAN came from the optimal or overfitting discriminator [32]. It is possible to use the diffusion process to

adaptively adjust the amount of information destroyed and the difficulty of the task for the discriminator based on how much it overfits. To estimate the overfitting as in the [5]

$$r_d = \mathbb{E}_{y,t\sim p(y,t)}[\text{sign}(\mathbf{D}(y,t) - 0.5)], \quad T = T + \text{sign}(r_d - d_{target}) \tag{21}$$

Where T is the adjustable maximum number of time-steps in the diffusion. r_d is calculated every 4 minibatches and T is updated accordingly. The timestep value t is sampled from p_π and based on the assumption that every time T is increased the discriminator is already good at lower t values, it would make sense to prioritize larger values of t to promote progressive learning.

$$t \sim p_\pi := \text{Discrete}(\frac{1}{\sum_{t=1}^{T} t}, \frac{2}{\sum_{t=1}^{T} t}, \ldots, \frac{T}{\sum_{t=1}^{T} t}) \tag{22}$$

In my version of Diffusion-StyleGAN2, I propose using Sinusoidal Positional Embedding [9] of time step t, instead of the simple learned embedding used in [5]. Sinusoidal positional embedding allows us to define the embedding using fixed functions rather than wasting the capacity of the network to learn the embedding implicitly. I argue that the continuous embeddings with smooth components are well suited for the model, as the maximum diffusion timestep T is constantly adjusted and learning the embeddings during training through a simple dense network would provide bias toward later timesteps. This comes from the priority distribution described earlier, which exposes the network to the non-uniform distribution of t, focusing more on higher values. For the revised architecture, see Figure 12 below.

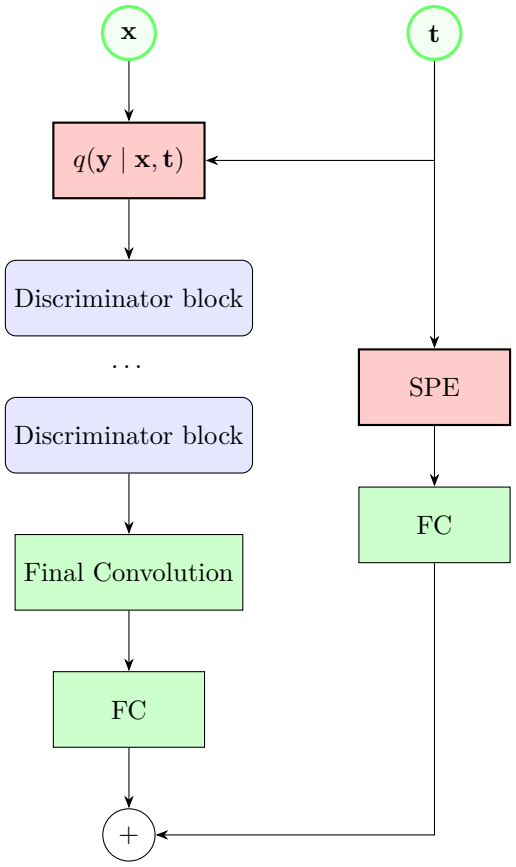


Figure 12: Proposed Diffusion-GAN discriminator with Sinusoidal Positional Embedding (SPE)

.

3.9.2 Training

Another change I have introduced to the models is applying diffusion only after a set number of training steps. This is analogous *to path length penalty* applied in StyleGAN2, which is only applied after certain number of training steps. The reasoning behind only applying diffusion after certain number of timesteps is to allow the discriminator to learn basic features faster and speed up convergence and then start perturbing the images to prevent overfitting. This technique is sound, as the model is only trained for a short amount of time before diffusion, not allowing enough time for the discriminator to overfit. It was also shown in the Diffusion-GAN paper that even after instance noise injection, the divergence between real and generated is minimized [5] and so it is valid to introduce perturbation later on in the training as the quantity being minimized is the same.

Hyperparameters The vanilla StyleGAN2 hyperparameters were kept the same. I have used linear noise scheduling with 300 timesteps with $\beta_{start} = 0.0001, \beta_{end} = 0.02$. This helps to keep the noise to a minimum at the start of training to allow the model to learn basic features first before adding perturbations. For STL10 the priority distribution was used as described earlier, while for CelebA and FashionMNIST showed better results with uniform distribution of diffusion.

4 Evaluation

In this section, I have evaluated the performance of the models by inspection, supported by the metrics implemented. I displayed and analysed the results obtained from training of the two core models and compared them to the results obtained by Diffusion-GAN and argued improvement in quality and stability. I have also shown that my version of Diffusion-GAN had higher precision, which correlates to image quality. I have also discussed the metrics used and when they are suitable and provide insight into performance.

4.1 Image quality

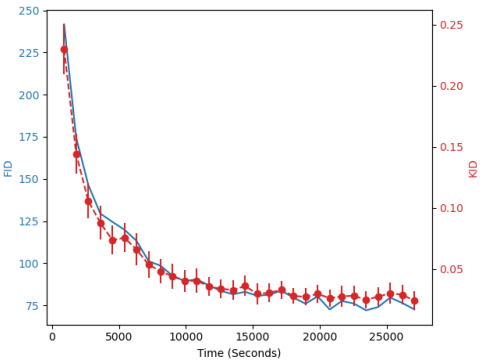
During training, I recorded various metrics at each epoch to track the image quality, image diversity, computation time and memory usage. To measure the InceptionV3-based metrics and capture the training process, the same set of 500 samples from the original dataset was used at each epoch, as picking new samples each time would introduce more variance in the results due to small sample size. StyleGAN2 had almost monotonically decreasing FID score at each epoch. This shows that StyleGAN2 is well suited for optimising perceptual image quality due to the mini-max nature of the discriminator and the generator. It had relatively low variance, despite only using 500 samples for FID calculation, which is considered relatively low. KID values were almost exactly correlated but scaled between $[0, 1]$ range, while also providing error, useful when dealing with limited number of samples. At the end of training, FID score was still relatively high. Several factors may have led to these inflated scores:

1. There could be a bias in the 500 images randomly picked from each dataset. If those 500 images are not representative of the entire dataset, then it may inflate the FID score.

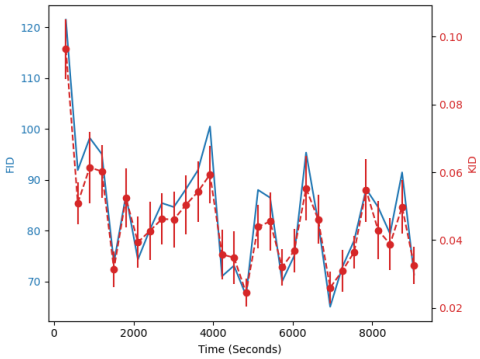
2. To calculate any Inception metric the generated images have to be scaled to 299x299 input for Inception model. Scaling using the built-in transform in Torchvision may cause aliasing, which can corrupt certain fine-grained features [47].

3. Training was only limited to 30 epochs and the score did not fully converge at the end.

Despite DDPM training requiring more steps to converge and having greater variation, as seen in Figure 13b, it achieved FID_{500} score below 70 after 5k wall-clock seconds, compared to 20k wall-clock seconds required by StyleGAN2 to achieve the same quality.



(a) StyleGAN2 training FID_{500} and KID_{500} on 64x64 CelebA dataset for 30 epochs. This displays stable training with fast convergence. Slower iterative reduction in FID per second past 75.



(b) DDPM training FID_{500} and KID_{500} on 64x64 CelebA dataset for 30 epochs. It is clear here that the training is a lot less stable due to the indirect ELBO loss. Training DDPM requires a lot more steps to converge.

(a) CelebA images produced by StyleGAN2 after 30 epochs and 27k wall-clock seconds of computation. $FID_{50k} = 16.15$.

(b) CelebA images produced by DDPM after 15 epochs and 5k wall-clock seconds of computation. $FID_{50k} = 31.75$.

Figures 14a and 14b show example generated images that both have an FID_{500} score ≈ 70. However, the diffusion images are substantially less realistic. Despite the fact that FID is supposed to correlate to perceptual image quality. A gray-shift artifact can be seen in Figure 14b that is produced by DDPM. This is caused by each color channel being treated independently, which caused the output channels to be averaged out. As stated previously, I used linear noise scheduling, which may destroy the contrast information too quickly at $T_{max} = 300$, making it harder to learn for the model. Another contributing factor which caused this artifact is the extensive use of attention in the model, as especially earlier in the training it encourages homogenized regions. [46].

After training on STL10 dataset, the models averaged lower KID and FID scores and lower perceptual quality by visual inspection. This was expected due to STL10's complexity [48]. As well as being multiclass, it does not guarantee center alignment of objects. Visual inspection of Figures 15a and 15b provides great insight into the inner workings of both

models.

(a) STL10 images produced by StyleGAN2 after 30 epochs and 6k wall-clock seconds of computation. $FID_{50k} = 46.7, IS_{50k} = 6.43$.

(b) STL10 images produced by DDPM after 30 epochs and 6k wall-clock seconds of computation. $FID_{50k} = 65.5, IS_{50k} = 6.12$.

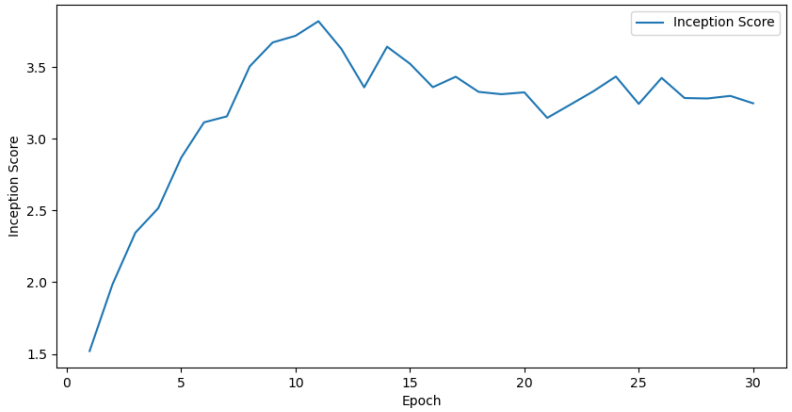
This shows a "blob" artifact in StyleGAN2. This is observed as all images share the same latent space $\mathcal{W}$. It is difficult for the network to learn the high-dimensional boundaries between classes as the model is not conditioned on class. This causes averaging, reducing the majority of shapes to blobs that lack resemblance to structures or animals. Indeed, a model explicitly conditioned on class labels produced drastically improved results [49]. In comparison, despite DDPM not being conditioned on image class, it performed significantly better, producing distinct shapes resembling different animals and structures, differentiating the different classes better.

4.1.1 Domain mismatch

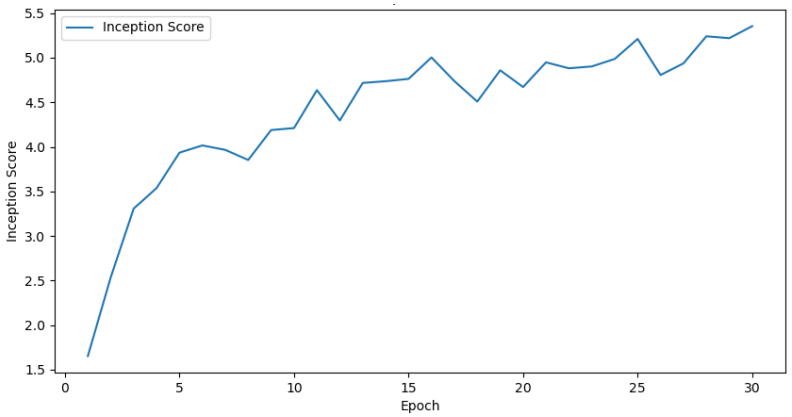
While the IS and FID are widely used for model evaluation, their utility becomes significantly reduced when there is a domain mismatch with the training dataset. For example, in the FashionMNIST dataset, there is a significant domain mismatch with the underlying model of all of these metrics. InceptionV3 was trained on the AlexNet dataset, which contains numerous classes, including some clothing. However, the overlap between the clothing classes of FashionMNIST and AlexNet is small. This results in a lower upper bound on the IS score; the metric now becomes more of a proxy of how well the generated images match AlexNet. Figure 16 shows images produced by the models trained on FashionMNIST. Despite being high fidelity, they scored worse on the conventional InceptionV3-based metrics compared to other natural datasets like STL10.

Figure 16: FashionMNIST images produced by DDPM after 30 epochs and 4k wall-clock seconds of computation. $IS_{50k} = 3.67$.

The effect of domain mismatch is further displayed in Figures 17a, 17b. Even though the StyleGAN2 trained on the FashionMNIST dataset (see Figure 16) scored a substantially lower IS score than the one trained on the STL10 dataset (see Figure 15a), the former ultimately had higher image fidelity.



(a) Recorded IS_{500} scores from StyleGAN2 trained on FashionMNIST dataset. Max IS_{500} reaching 3.8.



(b) Recorded IS_{500} scores from StyleGAN2 trained on STL10 dataset. Max IS_{500} reaching 5.4.

4.1.2 Latent space expressiveness

In order to assess the latent space improvements claimed through the inclusion of the *path length penalty* term in StyleGAN2, I performed the latent space interpolation experiment on both StyleGAN2 and DDPM as baseline, which does not introduce any improvements to its latent space. Figure 18 displays a clear transition from the leftmost image to the rightmost image. The transition involves gradual perceptual change in main features such as hair style, positioning and shade of skin.

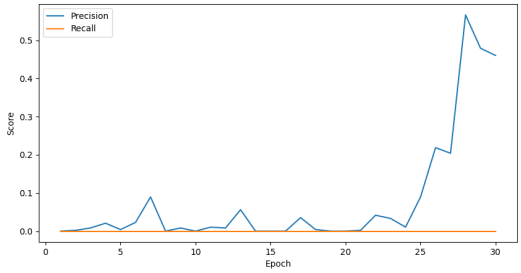
Figure 18: StyleGAN2 latent space interpolation, using algorithm described in Section 3.5.6, with α threshold set to 0.1 and 7 step.

Figure 19: DDPM latent space interpolation, using algorithm described in Section 3.5.6, with α threshold set to 0.1 and 7 step.

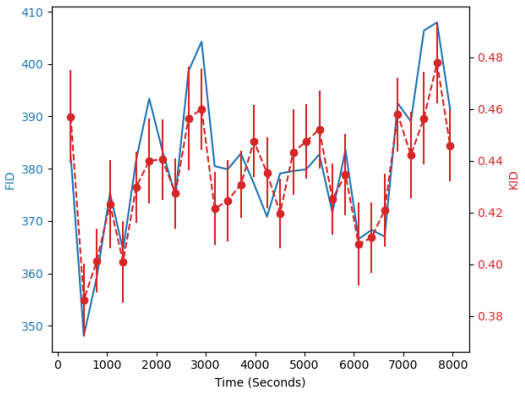
The interpolation was done by picking appropriate $w_1, w_2 \in \mathcal{W}$ and $z_1, z_2 \in \mathcal{Z}$, as described in the algorithm earlier and the DDPM baseline shows a very chaotic transition between the two points, with the interpolated pictures not having visual features in common.

4.2 Diversity

Although the three models showed varying diversity, mode coverage posed particular difficulties for GAN-like models. While the FID metric partially captured this, it was ultimately best shown through precision and recall values. During Diffusion-GAN training on the FashionMNIST dataset (see Figures 20a, 20b), FID and KID values remained high, indicating either mode collapse or that the model was not converging on a multi-class dataset. On the other hand, precision and recall values indicate near-zero zero diversity in generated images, as well as precision improvement over time, typical only of mode collapse. Clearly, FID and KID alone are insufficient metrics.



(a) Plot for precision (blue) and recall (orange) values.



(b) Plot of FID (blue) and KID (red) scores.

(c) Example images post-training.

Figure 20: Summary of Diffusion-GAN results for FashionMNIST.

Figure 20c shows that almost all generated samples were identical as "blobs" with averaged features. While this behavior is frequently observed in GANs [32], DDPM's objective to minimize the estimate for the KL divergence between the real and the generated distributions means that it directly encourages it to cover all modes of the data. Table 5 shows the results of training on a multiclass dataset FashionMNIST. The two Diffusion-GAN models achieved very low recall and low precision as they struggled to fit the non-continuous classes on $\mathcal{W}$ latent space. In the case of unconditional model, the path length penalty and smoothing may actually deteriorate the models ability to separate the classes. DDPM, while not having as expressive or smooth latent space, managed to capture a vastly greater range of

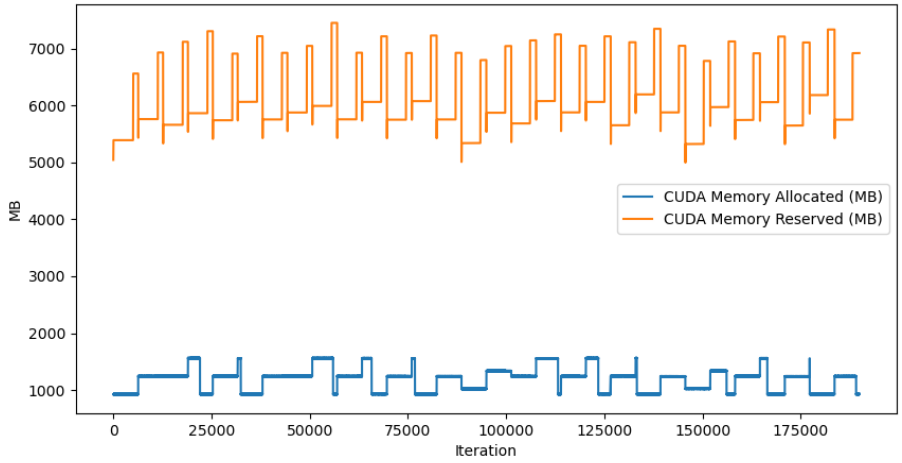
diversity.

Model	**Precision**$_{50k}$	**Recall**$_{50k}$
Diffusion-GAN	0.43	0.01
StyleGAN	0.41	0.04
DDPM	**0.67**	**0.54**

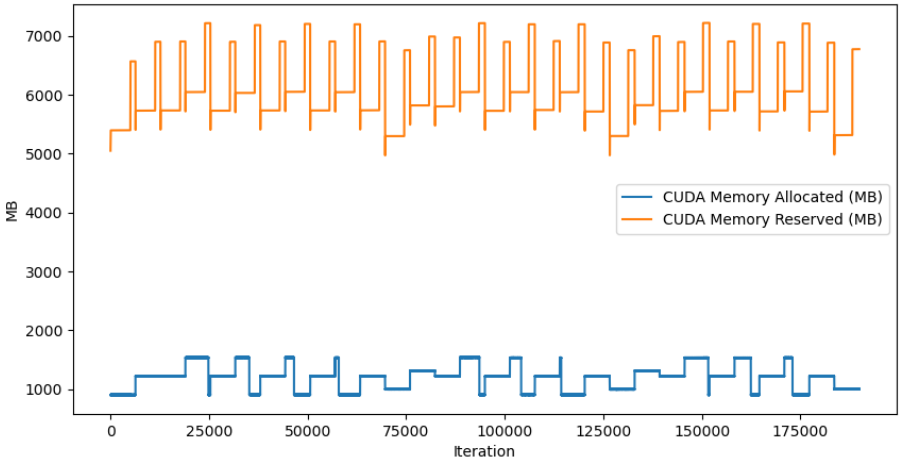
Table 5: Precision and Recall metrics for unconditional models trained on FashionMNIST dataset

4.3 Memory Usage and Computation Time

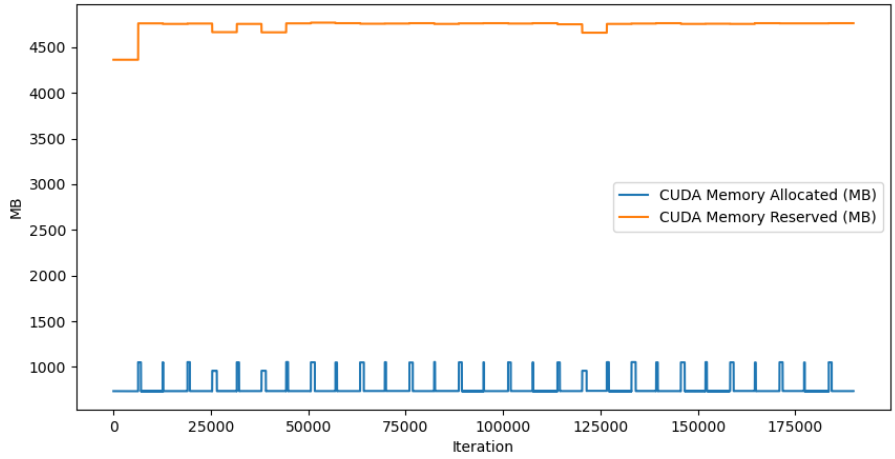
Memory usage and computation time are important factors to consider when training any generative AI model. All three models contained deep convolutional neural networks, which have a large number of parameters occupying space on vRAM memory. Additionally, DDPM required multiple passes through the network for inference, which greatly increased computation time. To measure the computation time taken to train the model, I used **wall-clock** seconds, as it includes the time CPU and GPU wait for IO and idle time, giving an unbiased and more realistic estimation of training time . All models were trained on GH200 superchip to allow for comparison between their training times. I tracked CUDA reserved memory for each model for each dataset. Figure 24 illustrates model memory usage when trained on the CelebA dataset. I kept the resolution and minibatch size constant to focus solely on model differences.



Diffusion-GAN



StyleGAN2



DDPM

Figure 24: CUDA Memory usage during training on 64x64 CelebA dataset with 32 samples minibatch size.

DDPM required less GPU vRAM, meaning that it scaled better with the increase in

resolution and batch size. StyleGAN2 and Diffusion-GAN may have had faster convergence for certain datasets, yet they typically required more GPU vRAM, which can be a limiting factor when training on high-resolution images. This is further supported by the calculated number of parameters in Table 6: DDPM has orders of magnitude fewer parameters. If memory cost becomes too high, it is necessary to reduce the minibatch size. This can further amplify the training instability [50]. It may also result in poor batch statistic calculations in layers such as minibatch standard deviation. This justifies using diffusion based models for large and high-resolution tasks and especially text-to-image, such as DALL-E 3 [51].

Model	**32×32**	**48×48**	**64×64**
StyleGAN2	8.9M	8.9M	18.2M
DDPM	260k	566k	989k
Diffusion-GAN	10.2M	10.2M	19.5M

Table 6: Total number of parameters of each model at different input resolutions. For StyleGAN2 and Diffusion-GAN. This calculation was derived by summing the number of parameters in the discriminator, generator and the mapping network.

4.4 Diffusion-GAN

Figure 25: CelebA images produced by Diffusion-StyleGAN2 + SPE after 30 epochs and 27k wall-clock seconds of computation.

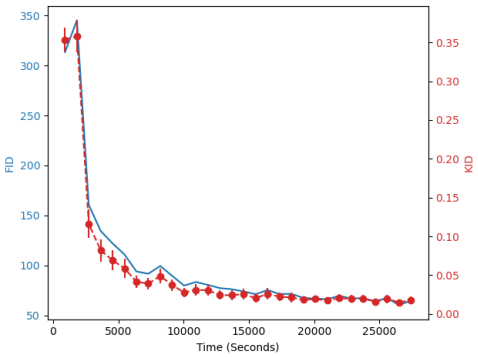


Figure 26: Diffusion-StyleGAN2 training FID_{500} and KID_{500} on 64x64 CelebA dataset for 30 epochs.

My implementation of the Diffusion-GAN model, was based on top of my StyleGAN2 combined with diffusion augmentation as described in [5] and my proposed Sinusoidal Positional Embedding (SPE) of time step t. It produced higher perceptual quality images by visual inspection and lower KID_{500} and FID_{500} scores than the vanilla StyleGAN2. Overall, the diffusion-injected discriminator produced more stable training with faster convergence, as indicated by the training graph in Figure 26, which shows a lower FID score at the same wall-clock time.

Model	**FID**$_{50k}$	**KID**$_{50k}$	**Precision**$_{50k}$	**Recall**$_{50k}$
StyleGAN2	**16.15**	**0.012** $\pm$ 0.0006	0.78	**0.11**
DDPM	31.75	0.03 $\pm$ 0.001	0.56	0.09
Diffusion-Gan	19.61	0.017 $\pm$ 0.0008	**0.86**	0.06

Table 7: Image quality and diversity metrics recorded for the three models trained on 64x64 CelebA.

Overall, the diffusion technique in GAN models improved image quality. However, FID does not accurately reflect this due to its conflation of diversity and quality. This is particularly problematic for GANs, which are typically optimised for high-fidelity image generation, even at the cost of reduced diversity (e.g. via the truncation trick) [20], [21], [2]. In such instances, precision provides a better insight into improvements in image quality, whereas recall can be used to monitor for mode collapse, as can be seen in Table 7. Additionally, both FID_{50k} and FID_{500} were computed for Diffusion-GAN and StyleGAN2: while Diffusion-GAN showed a lower FID_{500} score during training, after obtaining the final FID_{50k} score, StyleGAN2 managed to achieve a "better" result. This supports the earlier claim made in Section 3.5 that FID is inherently biased towards different models at different sample levels. Ultimately, without other data augmentation techniques, diffusion did not improve mode coverage. Nevertheless, it achieved high-fidelity samples and fast inference, so it may be useful for applications that require generating images on the fly.

5 Conclusions

5.1 Work completed

Overall, the project was successful, with all of the core and extension requirements completed. Most notably, my extension to the implementation of Diffusion-GAN produced high-fidelity images. After meeting my core and extension success criteria, I went beyond the original proposed plan and introduced new metrics to provide a better comparison and extended the models. I have implemented both of the core models, one of which was extended with added attention, while the other one was used as a base for the Diffusion-GAN extension model. All of the models were benchmarked on all three datasets and the results were provided in tabular form and discussed afterwards. The results served as motivation to implement the extension model and extend it further. Displaying and analysing the results also required developing a pipeline for operating on images and auxiliary scripts to display graphs, track metrics and combine images.

5.2 Future directions

A possible future direction would be to train my extended Diffusion-GAN until convergence and benchmark its performance to the original implementation. Furthermore, it could be fruitful to compare and benchmark diffusion to other image augmentation techniques such as the ones used in ADA version of StyleGAN2 [32], to see if there is a statistically significant difference in the recall of the images produced. With the emergence of ultra-high fidelity

generative AI models, I also believe that the community could benefit from more research into the development and implementation of more robust metrics. I also plan to extend the diffusion model further by adding improvements from the recent literature and attempt to develop a text-to-image model.

5.3 Lessons learned

I have found this project to be a fascinating exploration of modern generative AI. Throughout the project, I have applied my knowledge from Part IB and Part II courses and deepened that knowledge. It has been incredibly rewarding working on this project and I thoroughly enjoyed learning the theory behind the models. I acquired knowledge of stochastic processes, Markov chains and deep learning techniques. I also learned to run controlled experiments and benchmark generative AI models.

6 Bibliography

References

[1] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[2] Y. Alaluf, O. Patashnik, Z. Wu, A. Zamir, E. Shechtman, D. Lischinski, and D. Cohen-Or, "Third time's the charm? image and video editing with stylegan3," in *European Conference on Computer Vision*. Springer, 2022, pp. 204–220.

[3] A. Sauer, T. Karras, S. Laine, A. Geiger, and T. Aila, "Stylegan-t: Unlocking the power of gans for fast large-scale text-to-image synthesis," in *International conference on machine learning*. PMLR, 2023, pp. 30 105–30 118.

[4] E. Morales-Juarez and G. Fuentes-Pineda, "Efficient generative adversarial networks using linear additive-attention transformers," 2024. [Online]. Available: https://arxiv.org/abs/2401.09596

[5] Z. Wang, H. Zheng, P. He, W. Chen, and M. Zhou, "Diffusion-gan: Training gans with diffusion," 2023. [Online]. Available: https://arxiv.org/abs/2206.02262

[6] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," 2022. [Online]. Available: https://arxiv.org/abs/2010.02502

[7] Z. Xiao, K. Kreis, and A. Vahdat, "Tackling the generative learning trilemma with denoising diffusion gans," 2022. [Online]. Available: https://arxiv.org/abs/2112.07804

[8] T. Kynkäänniemi, T. Karras, S. Laine, J. Lehtinen, and T. Aila, "Improved precision and recall metric for assessing generative models," 2019. [Online]. Available: https://arxiv.org/abs/1904.06991

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[10] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[12] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," 2019. [Online]. Available: https://arxiv.org/abs/1809.11096

[13] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[14] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

[15] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International conference on machine learning*. PMLR, 2021, pp. 8162–8171.

[16] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[17] L. Weng, "What are diffusion models?" *lilianweng.github.io*, Jul 2021. [Online]. Available: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1501–1510.

[20] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[21] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 8110–8119.

[22] A. Buriro, F. Ricci, and B. Crispo, "Swipegan: swiping data augmentation using generative adversarial networks for smartphone user authentication," in *Proceedings of the 3rd ACM workshop on wireless security and machine learning*, 2021, pp. 85–90.

[23] L. Weng, "From gan to wgan," 2019. [Online]. Available: https://arxiv.org/abs/1904.08994

[24] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016. [Online]. Available: https://arxiv.org/abs/1511.06434

[25] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," 2018. [Online]. Available: https://arxiv.org/abs/1710.10196

[26] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017. [Online]. Available: https://arxiv.org/abs/1701.07875

[27] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016. [Online]. Available: https://arxiv.org/abs/1606.03498

[28] Lambda, "Gpu cloud – vms for deep learning," https://lambda.ai/service/gpu-cloud, n.d., accessed: 2025-01-12.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[30] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1708.07747

[31] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 215–223.

[32] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, "Training generative adversarial networks with limited data," *Advances in neural information processing systems*, vol. 33, pp. 12 104–12 114, 2020.

[33] S. Jayasumana, S. Ramalingam, A. Veit, D. Glasner, A. Chakrabarti, and S. Kumar, "Rethinking fid: Towards a better evaluation metric for image generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 9307–9315.

[34] M. J. Chong and D. Forsyth, "Effectively unbiased fid and inception score and where to find them," 2020. [Online]. Available: https://arxiv.org/abs/1911.07023

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[36] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton, "Demystifying mmd gans," 2021. [Online]. Available: https://arxiv.org/abs/1801.01401

[37] N. S. Detlefsen, J. Borovec, J. Schock, A. Harsh, T. Koker, L. D. Liello, D. Stancl, C. Quan, M. Grechkin, and W. Falcon, "Torchmetrics - measuring reproducibility in pytorch," 2022. [Online]. Available: https://www.pytorchlightning.ai

[38] A. Obukhov, M. Seitzer, P.-W. Wu, S. Zhydenko, J. Kyl, and E. Y.-J. Lin, "High-fidelity performance metrics for generative models in pytorch," 2020, version: 0.3.0, DOI: 10.5281/zenodo.4957738. [Online]. Available: https://github.com/toshas/torch-fidelity

[39] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," 2018. [Online]. Available: https://arxiv.org/abs/1706.08500

[40] M. Anderson, "Image synthesis sector has adopted a flawed metric, research claims," 2021. [Online]. Available: https://www.unite.ai/image-synthesis-sector-has-adopted-a-flawed-metric-research-claims/

[41] S. Bischoff, A. Darcher, M. Deistler, R. Gao, F. Gerken, M. Gloeckler, L. Haxel, J. Kapoor, J. K. Lappalainen, J. H. Macke, G. Moss, M. Pals, F. Pei, R. Rapp, A. E. Sağtekin, C. Schröder, A. Schulz, Z. Stefanidi, S. Toyota, L. Ulmer, and J. Vetter, "A practical guide to sample-based statistical distances for evaluating generative models in science," 2024. [Online]. Available: https://arxiv.org/abs/2403.12636

[42] labml.ai, "labml.ai deep learning paper implementations," https://github.com/labmlai/annotated_deep_learning_paper_implementations, 2024.

[43] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[44] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM journal on control and optimization*, vol. 30, no. 4, pp. 838–855, 1992.

[45] lucidrains, "Denoising diffusion probabilistic model, in pytorch," https://github.com/lucidrains/denoising-diffusion-pytorch, 2024.

[46] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International conference on machine learning*. PMLR, 2019, pp. 7354–7363.

[47] G. Parmar, R. Zhang, and J.-Y. Zhu, "On aliased resizing and surprising subtleties in gan evaluation," in *CVPR*, 2022.

[48] F. Branchaud-Charron, A. Achkar, and P.-M. Jodoin, "Spectral metric for dataset complexity assessment," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3215–3224.

[49] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *International conference on machine learning*. PMLR, 2017, pp. 2642–2651.

[50] A. Karnewar and O. Wang, "Msg-gan: Multi-scale gradients for generative adversarial networks," 2020. [Online]. Available: https://arxiv.org/abs/1903.06048

[51] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," 2021. [Online]. Available: https://arxiv.org/abs/2102.12092

[52] N. Klingler, "A complete walkthrough of convolution operations," Mar 2025. [Online]. Available: https://viso.ai/deep-learning/convolution-operations/

[53] K. Rana, "Pooling layer — short and simple." [Online]. Available: https://plainenglish.io/blog/pooling-layer-beginner-to-intermediate-fa0dbdce80eb

[54] M. Hollemans, "Upsampling in core ml." [Online]. Available: https://machinethink.net/blog/coreml-upsampling/

[55] A. Y. N. Maas and Hannun, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, 2013, workshop on Deep Learning for Audio, Speech and Language Processing.

[56] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[57] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

A Neural Network theory

A.1 Backpropagation

Backpropagation is a machine learning algorithm used for optimizing neural networks. It provides a method for computing the gradient of the loss function with respect to the weights of the networks. The optimizers then use these gradients to update the weights accordingly. More formally, the goal of backpropagation is to compute $\frac{\partial L}{\partial \theta_i}$ for each network parameter θ_i.

A.2 Convolutions

A convolution is a deep learning operation. It enables neural networks to efficiently capture spatial patterns within an image, such as edges, textures and shapes. They are used extensively within image detection and generation tasks due to high spatial correlation between pixels. This bias is then exploited. The convolution operation involves a kernel that slides across the image, computing a dot product between the kernel and the overlapping region of the input at each location. A convolution is defined by:

- **Kernel size**, i.e 3x3, 4x4, ... The default kernel size for most applications is 3x3.
- **Stride** - indicates the distance the kernel slides by after each calculation. The default stride for most applications is 1 pixel.

Figure 27 shows a 3x3 kernel with a stride of 1. After computing the value as shown, it will shift by one pixel to the right and repeat.

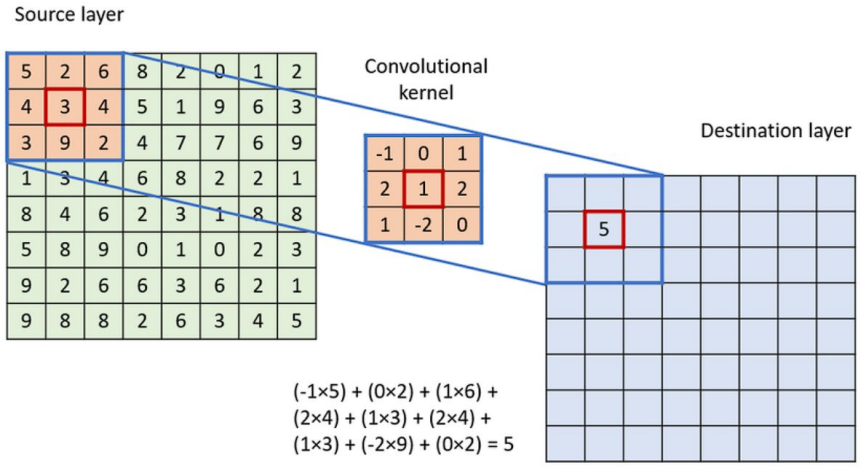


Figure 27: Taken from [52]. Shows the computation of a single pixel during a kernel pass.

A.3 Upsampling and downsampling images

In neural networks, upsampling and downsampling play a crucial role in changing the spatial resolution of an image. This can be done to extract lower/higher level features. Decreasing spatial resolution also decreases computational load, while increasing spatial resolution is usually down in segmentation or image generation tasks, where the output resolution needs to match the input. There are multiple ways to downsample or upsample an image, although the ones used in this project were **max pooling** downsampling and **bilinear** upsampling.

A.3.1 Max pooling

This is one of the implementations of downsampling, which works similar to a convolution operation. A sliding kernel is used on the image to calculate the value of a pixel, similarly to convolution. In contrast to normal convolution, it computes the maximum value, instead of the dot product. To avoid aliasing due to overlap between kernels, the stride is set the size of the kernel so each pass is independent, as shown in Figure 28.

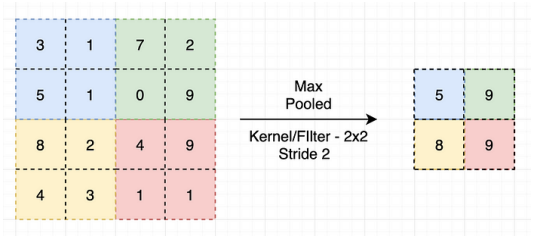


Figure 28: Taken from [53]. Shows a 2x2 max pool kernel applied to a 4x4 image to reduce its spatial resolution in half.

A.3.2 Bilinear upsampling

Bilinear upsampling is a particular implementation of upsampling. It increases the spatial resolution by linearly interpolating the values of the four nearest pixels to calculate the new pixel value.

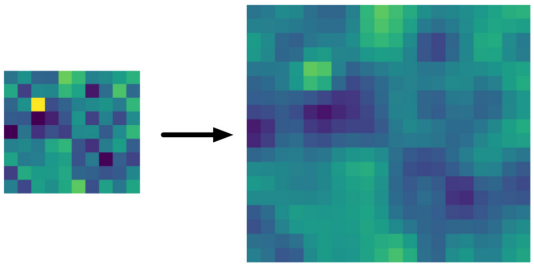


Figure 29: Taken from [54]. Shows bilinear upsampling, doubling the spatial resolution of an image.

A.4 Activation functions

Rectified Linear Unit (ReLU) is a widely used activation function s.t $f(x) = max(0, x)$. Throughout this project, I have used a modified version of it called LeakyReLU. It was proposed as an improvement to ReLU, aimed at tackling the *dying neuron problem* [55].

Instead of reducing any negative input to 0, LeakyReLU scales it by a very small constant, aimed at improving the gradient flow and preventing the neurons from dying.

Definition

The **dying neuron problem** occurs in networks using ReLU activation, when a neuron's weights are updated such that its input becomes negative for all future inputs. The gradient through that neuron then becomes zero, thus it stops learning and becomes *dead*. This reduces the models learning capacity.

Another version that was proposed more recently is Sigmoid Linear Unit, s.t

$$f(x) = \sigma(x) \cdot x \quad \text{where } \sigma(x) = \frac{1}{1 + e^{-x}} \tag{23}$$

It was shown to improve the gradient flow and performed better than ReLU and LeakyReLU in deep neural net architectures such as EffcientNet [56], [57].

B MMD

Maximum Mean Discrepancy (MMD) is a non-parametric statistical test used to compare whether two distributions P and Q are the same based on samples from each.

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

Where $k(x, y)$ is a *kernel function*, which measures the similarity between two inputs by computing the inner product of their higher-dimensional representation.

C Gallery

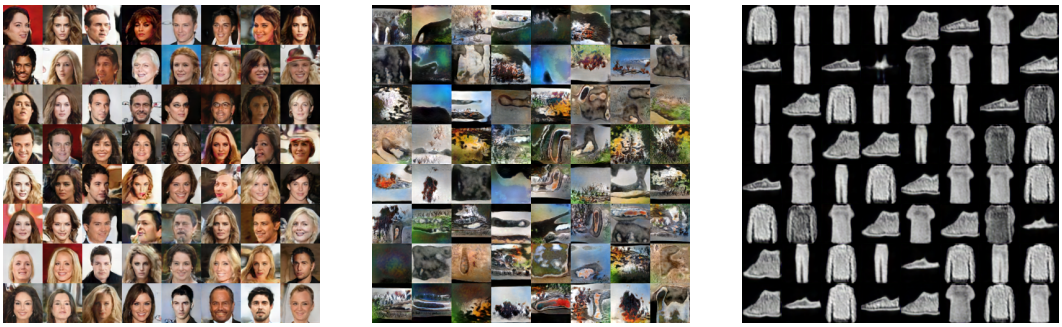
Showcase of images produced by the models

(a) (b) (c)

Figure 30: DDPM generated images 64x64 resolution. (a) CelebA; (b) STL10; (c) Fashion-MNIST

(a) (b) (c)

Figure 31: StyleGAN2 generated images 64x64 resolution. (a) CelebA; (b) STL10; (c) FashionMNIST



(a) (b) (c)

Figure 32: Diffusion-GAN generated images 64x64 resolution. (a) CelebA; (b) STL10; (c) FashionMNIST

D Hyperparameters

Listing 1: StyleGAN2 CelebA Hyperparameters

```
model: stylegan2
dataset: CelebA
res: 64
batch_size: 32
channels: 3
dim_w: 512
mappingnet_layers: 8
epochs: 30
lr: 1e-3
mapping_lr: 1e-5
mixing_prob: 0.9
grad_pen_interval: 32
grad_pen_coef: 10
path_pen_interval: 32
path_pen_after: 5000
```

```
device: cuda
save_dir: ./results/stylegan/
```

Listing 2: StyleGAN2 STL10 Hyperparameters

```
model: stylegan2
dataset: STL10
res: 64
batch_size: 32
channels: 3
dim_w: 512
mappingnet_layers: 8
epochs: 30
lr: 1e-3
mapping_lr: 1e-5
mixing_prob: 0.9
grad_pen_interval: 32
grad_pen_coef: 10
path_pen_interval: 32
path_pen_after: 5000
device: cuda
save_dir: ./results/stylegan/
```

Listing 3: StyleGAN2 FashionMNIST Hyperparameters

```
model: stylegan2
dataset: FashionMNIST
res: 32
batch_size: 32
channels: 3
dim_w: 512
mappingnet_layers: 8
epochs: 30
lr: 1e-3
mapping_lr: 1e-5
mixing_prob: 0.9
grad_pen_interval: 32
grad_pen_coef: 10
path_pen_interval: 32
path_pen_after: 5000
device: cuda
save_dir: ./results/stylegan/
```

Listing 4: DDPM CelebA Hyperparameters

```
model: ddpm
dataset: CelebA
res: 64
```

```
batch_size: 32
channels: 3
dim_mults: [1,2,4]
epochs: 30
lr: 1e-3
timesteps: 300
device: cuda
save_dir: ./results/ddpm/
```

Listing 5: DDPM STL10 Hyperparameters

```
model: ddpm
dataset: STL10
res: 64
batch_size: 32
channels: 3
dim_mults: [1,2,4]
epochs: 30
lr: 1e-3
timesteps: 300
device: cuda
save_dir: ./results/ddpm/
```

Listing 6: DDPM FashionMNIST Hyperparameters

```
model: ddpm
dataset: FashionMNIST
res: 64
batch_size: 32
channels: 1
dim_mults: [1,2,4]
epochs: 30
lr: 1e-3
timesteps: 300
device: cuda
save_dir: ./results/ddpm/
```

Listing 7: Diffusion-GAN CelebA Hyperparameters

```
model: diffusion_gan
dataset: CelebA
res: 64
batch_size: 32
channels: 3
dim_w: 512
mappingnet_layers: 8
epochs: 30
lr: 1e-3
```

```
mapping_lr: 1e-5
mixing_prob: 0.9
grad_pen_interval: 32
grad_pen_coef: 10
path_pen_interval: 32
path_pen_after: 5000
t_update_interval: 4
d_target: 0.6
update_kimg: 100
device: cuda
save_dir: ./results/diff_gan/
```

E Proposal

Dmytro Mai Phase 3

dm933

October 2024

1 Proposal

Supervisor: Chengliang Zhou UTO: Cengiz Oztireli

1.1 Title

A thorough comparison/analysis/summary of the pros & cons of the diffusion model and GAN

1.2 Description

The project aims to provide a comprehensive comparison of two generative machine learning models: General Adversarial Network (GAN) and Diffusion model. GANs are able to produce realistic images, and replicating and generalising well however, they suffer from mode collapse and are unstable during training. Diffusion models on the other hand mitigate those disadvantages at the cost of higher computational requirements. The project will delve into architecture, real world applications and use cases, and a comparison of their advantages and disadvantages. As an extension the aim is replicate and implement new technologies like patch-diffusion or GAN-diffusion models and thoroughly analyse them.

1.3 Starting point and motivation

Recent developments in generative models such as Generative Adversarial Network and Diffusion models have demonstrated exciting results in image generation. Each model has trade-offs in stability, use of computational resources and accuracy. GANs are known for their ability to produce highly realistic images, yet they suffer from mode collapse and unstable training. Diffusion models, on the other hand, offer more stability and diversity in generated images but come with higher computational costs. The comparison is a crucial step in addressing key challenges in generative modelling. The extension of this project aims to dive into building hybrid models for specialised purposes or limited computational resources, without sacrificing too much quality or diversity in generated images.

1.4 Data

It is crucial that the models are trained on the same dataset as that will provide an important comparison point and benchmark for testing them. Here are some possible publicly available datasets:

- MNIST - small grayscale handwritten digits. This dataset is good for some basic comparison and would not require large computational resources to train on, while providing some useful insights.
- CIFAR-10 - it is 32x32 colour dataset of 10 different classes of objects. This is higher resolution than the previous.
- CELEB-A (subset) - contains color pictures of different celebrity faces. This can be much higher resolution than other two datasets and will be useful for testing mode-collapse of GAN network.

It is likely that one or more of the above datasets will be used however, dataset will be picked according to the computational resources available and suitability for clear comparison.

1.4.1 Evaluation

As metrics to assess the models in my project and used as success criteria I will use the following:

Generated Image quality:

- Fréchet inception distance (FID) is a metric for quantifying the realism and diversity of images generated by generative adversarial networks (GANs).
- The Inception Score (IS) is an algorithm used to assess the quality of images created by a generative image model such as a generative adversarial network (GAN). The score is calculated based on the output of a separate, pretrained Inceptionv3 image classification model applied to a sample of (typically around 30,000) images generated by the generative model.

Training stability:

- I will monitor the convergence behavior and occurrences of mode collapse, comparing it to the training process of a diffusion model. Loss curves are usually sufficient but Evidence Lower Bound (ELBO) may be used.
- Time to convergence and training cost will be used as baselines to evaluate computational efficiency and stability.

Computation cost:

- For Diffusion models, I will analyze memory consumption, model size, and training time, comparing them to GANs under similar conditions.

Extension metric

- I will use DCGAN/StyleGAN2 for GAN models and DDPM diffusion model will be used as the baselines.
- Patch diffusion and GAN-Diffusion will be attempted as an extension. They will be compared to the baseline models previously mentioned to inspect improvements or trade-offs.

1.5 Extension

For the extension of the project I would like to attempt to recreate the Diffusion-StyleGAN2 model, which combines the two previously compared models and attempts to take the best of both worlds. I would like to then investigate how that changes the training process such as computational resources required, accuracy such as FID and IS scores, and the impact it has, and where this technology is applicable.

1.6 Timeline:

*WP- Work Package (2 weeks)

WP1	Literature review
WP2	Set up the environment and obtain the datasets
WP3	Implement Diffusion model
WP4	Implement GAN model
WP5	Progress report and eval
WP6	Compare and analyse results
WP7	Attempt to extend the models (patch diffusion)/work on Diffusion-GAN
WP8	Final tests and comparing findings
WP9	Filling in the gaps (results) or working on Extension
WP10	Finishing write up