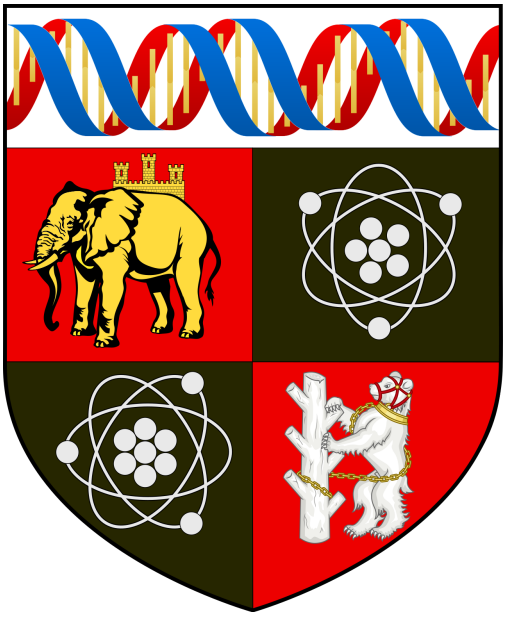
A Comparative Analysis of Generative Paradigms for Text-to-Image Synthesis



Author: Samir Rajesh

Supervisor: Dr. Fayyaz ul Amir Afsar Minhas

Year of Study: 3

Department of Computer Science

University of Warwick

April 30, 2024

Abstract

We present a comparative analysis of two generative paradigms for image synthesis: Diffusion Models and Generative Adversarial Networks. We dissect the architectural complexities of these models and develop an understanding of their functionality in image synthesis. We use both quantitative and qualitative methodologies in our analyses, measuring image fidelity using Fréchet Inception Distance. According to our findings, unconditional diffusion attained an FID score of 26.14, and conditional diffusion improved to 21.11. Meanwhile, unconditional and conditional GANs produced scores of 14.76 and 11.46, respectively, representing a disparity from recent literature where DDPMs typically outperform GANs, prompting further investigation. We developed the foundational models—an unconditional diffusion model and an unconditional deep convolutional GAN—extending them to class-conditioned models. We find that conditional generation notably enhanced image quality. A significant achievement of this work is the development, training, and the gained understanding of generative paradigms, facilitating a discussion of a hybrid generative framework that harnesses the strengths of both models. Despite computational constraints, which limited the complexity and depth of model testing, our thesis provides a comprehensive dive into the theory and development of generative paradigms. We establish a foundation for future work to bridge the gap observed in performance metrics.

Keywords: Diffusion Models, Generative Adversarial Networks, Convolutional Neural Networks, Image synthesis, Machine Learning, Deep Learning, Comparative analysis, Python

Acknowledgements

I would like to express my sincerest gratitude to Dr. Fayyaz Minhas, my project supervisor, for his steadfast support and guidance throughout the course of this project. His understanding and accommodation during my medical leave from university, as well as his advice regarding machine learning techniques, were invaluable to the continuation and completion of our research.

Special thanks are due to Dr. Rossella Suma, my second assessor, whose insightful questions and constructive feedback proved to be instrumental in identifying gaps in our research and raising the overall calibre of our work.

Finally, I remain profoundly grateful to my family and peers for their unconditional support and belief in my abilities. They have been a source of motivation throughout this academic endeavor.

Contents

List of Figures

List of Algorithms

List of Tables

Acronyms

This section provides a list of acronyms and abbreviations used throughout this thesis. Familiarizing yourself with these terms prior to reading will enhance your understanding of the material discussed within. The acronyms are presented alongside their full form to ensure clarity, regardless of the reader's prior familiarity with the topics covered.

Table 1: List of Acronyms

Acronym	**Description**
GAN	Generative Adversarial Network
DCGAN	Deep Convolutional Generative Adversarial Network
cGAN	Conditional Generative Adversarial Network
cDCGAN	Conditional Deep Convolutional Generative Adversarial Network
DDPM	Denoising Diffusion Probabilistic Model (synonymous with Diffusion Model)
FID	Fréchet Inception Distance
MSE	Mean Squared Error
BCE	Binary Cross Entropy
MLP	Multi-Layer Perceptron
(Leaky) ReLU	(Leaky) Rectified Linear Unit
SiLU	Sigmoid Linear Unit
CNN	Convolutional Neural Network

1 Introduction

1.1 Background

Generative models are frameworks capable of creating new content based on learned data patterns. At the core of this emerging technology, are two innovative approaches that have recently gained massive prevalence in the industry: Generative Adversarial Networks (GANs) and Diffusion Models (DDPMs), both of which have significantly contributed to the field of synthetic image generation.

GANs, conceptualized by Goodfellow et al in 2014 [1], are composed of dual neural networks trained in an adversarial manner. A generator network creates images from noise attempting to replicate the original data distribution from the training data, while a discriminator network attempts to classify data as either authentic (belonging to the original dataset) or synthetic (produced by the generator). Goodfellow et al. show that this adversarial training process, depicted in Figure 1, drives the generator to continuously refine its capabilities, resulting in a network that produces realistic synthetic images [1].

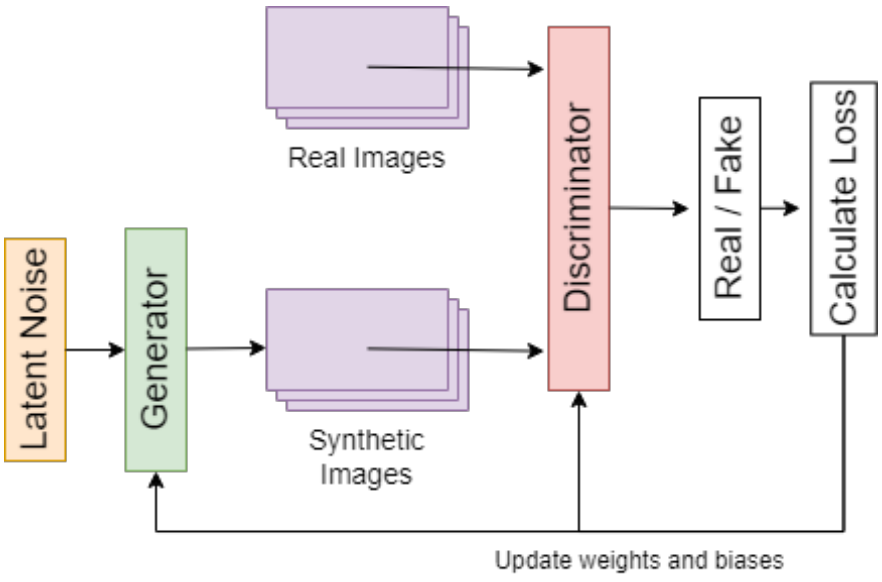


Figure 1: Generative Adversarial Network Architecture

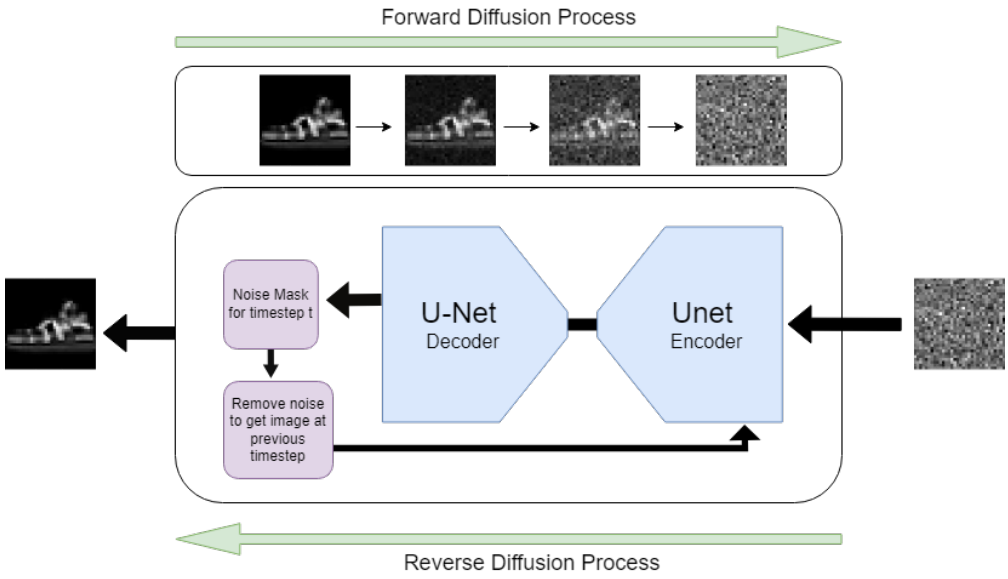


Figure 2: Diffusion Model Architecture

Diffusion models, initially conceptualized by Sohl-Dickstein et al. in their paper 'Deep Unsupervised Learning using Nonequilibrium Thermodynamics' [2], offer a novel approach for data synthesis. They introduce noise into real data, and then attempt to iteratively reverse this process. Essentially, in an attempt to reconstruct the original data by predicting and removing the added noise, they generate new data that resembles the training data. The diffusion architecture is illustrated in Figure 2.

While the low-level technical details of these models are complex, the high-level background provided here establishes a foundation for a more detailed exploration that follows in the Literature Review (Section 2).

Comparative analyses of diffusion models and GANs has been previously undertaken by researchers such as Dhariwal et al.[3]. However, these studies have predominantly utilized state-of-the-art models with significant computational resources while this project addresses the challenge of replicating such comparative analyses under computational constraints, specifically within the limitations of a single-GPU setup. This constraint represents a common reality for many developers in the field, making the findings of this study particularly

relevant to those working without access to an extensive computational infrastructure. We also utilize Classifier-Free Guidance in our conditional DDPM, as opposed to a classifier guided DDPM used by Dhariwal et al.[3].

1.2 Motivations

The primary motivation behind this study stems from a twofold desire: firstly, we wish to gain a practical understanding of the architecture and development of generative models, and secondly, to conduct an independent comparative analysis of these models that stands on its own merit against existing research. Our research aims to elucidate the process of developing Diffusion Models and GANs for individuals without access to high-end computational resources, thereby contributing practical insights to the broader AI community. The comparative results obtained from this project are intended to complement the existing literature and serve as a benchmark for what can be achieved under hardware constraints, bridging the gap between high and low-resource computational settings.

1.3 Project Objectives, Scope, Limitations

1.3.1 Objectives

The objectives of this study are outlined as follows:

- **Model Development:** Develop the foundational unconditional diffusion model and deep convolutional GAN.
- **Conditional Generation:** Extend our foundational models with class conditioning, employing Classifier-Free Guidance [4] for Diffusion Models and integrating label embeddings for the Conditional Deep Convolutional GAN.
- **Quantitative Analysis:** Conduct an analysis using Fréchet Inception Distance (FID) [5], to assess our models' strengths and weaknesses in generating images from a quantitative perspective.

- **Qualitative Analysis:** Conduct a visual review of the models' generated images, and detail the nuanced differences in their samples.
- **Comprehensive Overview and Future Directions:** Compile a thorough summary of the developed models' architectures and results, providing insights into potential future enhancements and concluding with implications for further research in the field.

1.3.2 Scope

This study is scoped within the following parameters:

- **Data Use:** We train on the FashionMNIST dataset [6], which provides an elementary and standardized environment for model training and evaluation.
- **Focus on Development and Analysis:** We focus on the development process and comparative performance analysis of generative models within our limited computational framework.
- **Performance Metrics:** We employ FID scores as the primary quantitative evaluation metric to measure image synthesis quality, recognizing its widespread use in the field.

1.3.3 Limitations

The following limitations are acknowledged in this study:

- **Computational Resources and Model Complexity:** Our single-GPU setup limits the complexity and diversity of the models that can be developed as well as the rigor with which they can be tested.
- **Data Complexity:** Our reliance on the FashionMNIST dataset [6] may not fully represent the challenges posed by more complex and diverse datasets. The choice of this dataset is justified in the Data Strategy (Section 3.1) while also elaborating on its negative aspects.
- **Scope of Application:** Our research is focused on image synthesis and does not explore the full spectrum and potential applications of generative models.

1.4 Thesis Structure

In conclusion, our research embarks on a comparative exploration of developing and training Diffusion Models and GANs within the bounds of a single-GPU environment. The following sections will dissect and expand upon the various facets of our research, laying the groundwork for a deep understanding of generative models.

We provide an in-depth examination of the seminal research on generative models, and discuss important concepts required to understand our implementations in the Literature Review (Section 2). The Methodology (Section 3) will outline the systematic approach we adopted to develop and test the models, detailing the dataset selection, and model development strategy employed. In the Design (Section 4) and Implementation (Section 5), we discuss the architectural choices, algorithms, and processes behind model construction in detail. Testing and Evaluation (Section 6) describes the tests we conducted on our models, followed by an extensive quantitative and qualitative evaluation of our models' results. Finally, the dissertation concludes with a synthesis of the findings and an acknowledgment of the limitations, complemented by a discussion of potential avenues for future research in Conclusions and Future Work (Section 7). For readers seeking additional insights into how the project was structured and managed, we includes a Project Management section (Section 8). Finally, Author's Reflections (Section 9) provides personal insights on the research journey and its impact on our understanding of the field.

2 Background and Literature Review

In this literature review, we delve into the background, specifics, and theory of Generative Adversarial Networks (GANs) and diffusion models. We provide a detailed exploration of their theory, supported by the research that shaped their creation and subsequent improvements. We also examine the relevant research in benchmarking methods commonly used to evaluate these models, as well as previously conducted comparative analyses of these models. In doing so, we set the stage for the research and development undertaken in this thesis.

2.1 Generative Adversarial Networks

2.1.1 Foundations of Generative Adversarial Networks

Generative Adversarial Networks (GANs), as detailed in the seminal work by Goodfellow et al.[1], are a novel framework for generative modeling that leverage an adversarial process. The architecture comprises of two neural networks: the generator, which synthesizes data, and the discriminator, which evaluates their authenticity [1]. The generator is trained to maximize the probability that the discriminator makes a mistake in its classification, forming a "minimax two-player game" dynamic within the system [1].

The practical advantage of GANs is their adversarial training methodology, which uses backpropagation and avoids the complications of Markov chains [1]. Goodfellow et al. asserted through empirical evidence that, assuming the implementation of a stable training architecture, GANs are theoretically capable of replicating any data distribution [1].

However, despite this theoretical promise, the application of GANs involves various iterative and numerical optimization techniques to balance their adversarial dynamics [1]. This training process must be finely tuned to prevent either network from overpowering the other, and to maintain a competitive yet productive equilibrium, where both network's losses converge.

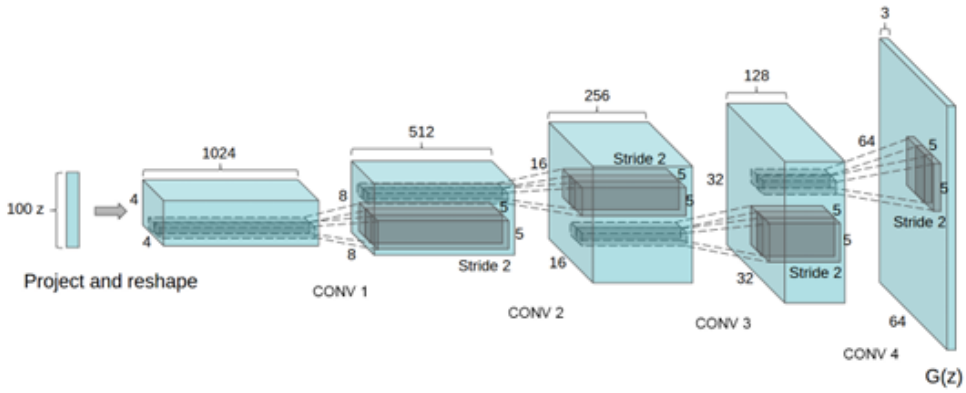


Figure 3: DCGAN Generator architecture showing a 100-dimensional uniform distribution Z projected into a 64 x 64 pixel image through a series of four fractionally-strided convolutions. Figure 1 from [7]

2.1.2 Deep Convolutional GAN

The introduction of GANs made a substantial impact in the field of generative modelling, sparking a multitude of enhancements in subsequent research with the aim of improving their stability and utility across various domains. Radford et al. built upon the original GAN concept by introducing a novel class of Convolutional Neural Networks (CNNs) named Deep Convolutional Generative Adversarial Networks (DCGANs) [7], presenting a leap in the stable training and sampling quality of GANs.

The convolutional architecture of DCGANs is a crucial factor in their success. Radford et al. established DCGANs and introduced several key architectural innovations that enhanced the model's performance and stability [7]. They replaced deterministic spatial pooling with strided convolutions in the discriminator and fractional-strided convolutions in the generator, promoting the learning of spatial downsampling and upsampling [7]. The elimination of fully connected layers further solidified the model's robustness, contributing to improved training convergence. A critical component was the use of Batch Normalization, which normalized each unit's inputs to have zero mean and unit variance, effectively addressing training issues related to initialization and encouraging gradient flow in deeper models. However, the indiscriminate application of batch normalization could lead to oscillation and instability, which was mitigated by excluding it from the generator's output layer and the discriminator's input

layers. Architecturally, the use of ReLU activation in the generator (except for the output layer where Tanh is used) and LeakyReLU in the discriminator marked a strategic departure from the original GAN paper's choice of maxout activations. These design choices established by Radford et al.[7] demonstrate that specific architectural choices are instrumental in implementing a stable training procedure and improving the quality of generated samples in GANs.[7]

The DCGAN model is a significant advancement not only in generative modeling but also in unsupervised learning, where it serves as a powerful feature extractor for subsequent supervised tasks[7]. An example of a DCGAN generator architecture from Radford et al.'s paper [7] is included in Figure 3, providing a clear depiction of the DCGAN generator, which upsamples a latent noise vector z through fractionally strided convolutions. The DCGAN discriminator works similarly in the opposite way, using strided convolutions to downsample an image until it can be passed through a fully connected layer with a SiLU activation to map it to a probability for whether the input image is real or synthetic.

2.2 Diffusion Models

2.2.1 Foundations of Diffusion Models

Diffusion models were developed as a class of generative models capable of producing high-quality samples through learning to reverse a gradual noising process. The theoretical foundation of diffusion models was laid by Sohl-Dickstein et al. in their 2015 paper, "Deep Unsupervised Learning using Nonequilibrium Thermodynamics," [2], which systematically destroyed structure in data distributions through a diffusion process and learned a reverse diffusion process to restore that structure.

Jonathan Ho et al.'s paper "Denoising Diffusion Probabilistic Models" [8] expanded upon this foundational study, outlining a framework in which data distribution is gradually corrupted by Gaussian noise over a sequence of time steps. The forward diffusion process, is characterized by its Markovian nature, visible in Figure 4 where $q(x_t|x_{t-1})$ shows that an

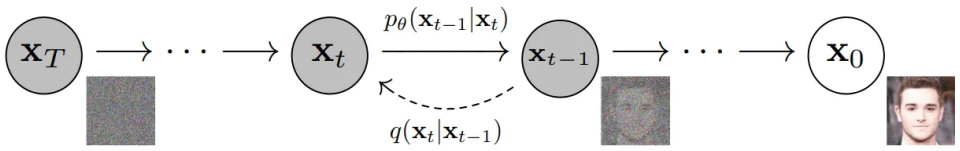


Figure 4: DDPM's structure depicted as a directed graphical model (Figure 2 from [8])

image x_t at timestep t depends directly on the image at the previous noising timestep. It also demonstrates the process's ability to produce latent noise samples x_1 through x_T by adding Gaussian noise with a predefined variance schedule β_t [8]. Ho et al. outline the process by which they incrementally convert a data sample to a latent variable where x_T approaches an isotropic Gaussian distribution in their DDPM paper [8]. The reverse process involves a generative model parameterized by a neural network that learns to denoise these latents, essentially reversing the forward process to generate a sample resembling the training data distribution [8].

A significant innovative suggestion introduced by Ho et al.[8] is the utilization of the U-Net architecture within the reverse diffusion process. The U-Net, originally introduced by Ronneberger et al.[9], is comprised of an encoder-decoder structure connected by a bottleneck, with skip connections that concatenate features from the encoder to the decoder at equal levels, enhancing the network's ability to capture fine-grained details in the output. The initial architecture proposed by Ronneberger et al. is shown in Figure 5. Ho et al.[8] adapt this U-Net architecture following the "structural backbone of an unmasked PixelCNN++[10, 8]", based on a Wide ResNet. This modified architecture incorporates several key adaptations to suit the requirements of diffusion models.

Their architecture employs group normalization instead of weight normalization, with the aim of simplifying the implementation [8]. They designed multiple networks for different resolutions, each containing two convolutional operations at each residual block. Notably, self-attention blocks are utilized between the convolutional blocks at the 16×16 resolution, enhancing the model's ability to dynamically focus on relevant features during the diffusion process.[8]

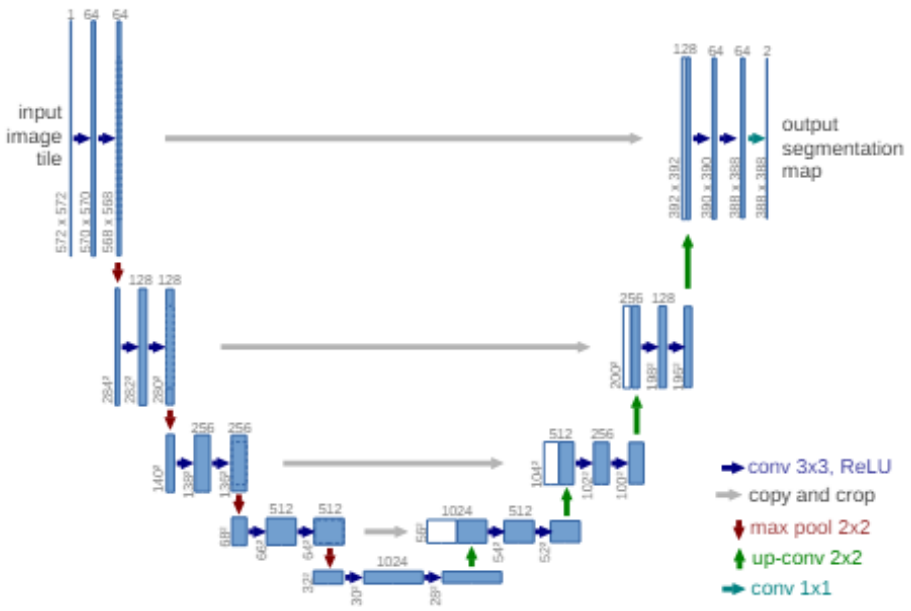


Figure 5: Example of a U-net architecture from Figure 5 of Ronneberger et al.'s paper introducing the U-Net [9]

While the discussion of utilizing a Wide ResNet, group and weight normalization, and the use of self-attention blocks presents interesting avenues for enhancing model performance, a detailed exploration of these technical choices is outside the scope of this project. This dissertation focuses on the core aspects of the diffusion model architecture as they directly pertain to the implementation carried out, leaving the in-depth examination of these advanced topics to future work. We however do utilize the convolutional operations within each Residual Block as well as the sinusoidal positional embedding proposed by Ho et al.[8] detailed below.

Ho et al.[8] propose the integration of the diffusion timestep t into every residual block in the network by embedding a Transformer sinusoidal position embedding. This integration allows the model to adapt its processing based on the temporal stage of the diffusion, a feature implemented within our U-Net architecture.

This architecture is particularly effective in the learning of Gaussian noise applications in

the reverse diffusion process. At each reverse step, the model predicts the noise that was added to the image at that timestep during the forward diffusion process [8]. This approach enables the reverse diffusion process to gradually and effectively learn to reconstruct data that ideally resembles the original data samples.

2.2.2 Advancements in Diffusion Models

Subsequent improvements by Nichol et al. in their paper titled "Improved Denoising Diffusion Probabilistic Models" [11] addressed the issue of log-likelihood optimization, a key performance metric for generative models. By introducing modifications such as a cosine variance schedule, learning the variances of the reverse diffusion process, and utilizing a hybrid training objective that combines the variational lower bound (VLB) with a simplified objective, the Improved Denoising Diffusion Probabilistic Models (Improved DDPMs)[11] not only maintained high sample quality but also achieved competitive log-likelihoods even on diverse datasets like ImageNet. Additionally, Nichol et al.[11] enhanced the practicality of these models by significantly reducing the number of necessary sampling steps for image generation (a product of learned variances), thereby accelerating the sampling process to make diffusion models more applicable in real-world scenarios. This improvement in efficiency is particularly relevant for practical applications where computational resources are a constraint. However, due to time constraints, not all the improvements from their paper were able to be incorporated in our implementation of the DDPM, and are left to future work.

These advancements positioned diffusion models as highly competitive within the field of generative modeling, offering a blend of high-quality output and efficient computation showing promise in scalability with increased training resources. The robustness of diffusion models as well as their capacity for growth, demonstrated through their improved log-likelihoods and sample quality [11], indicate a fertile ground for future research in generative modeling.

2.3 Guidance Techniques

2.3.1 Classifier-Free Guidance

Classifier-Free Guidance in diffusion models, as introduced by Ho and Salimans in their paper titled "Classifier-Free Diffusion Guidance" [4], enables us to condition our model on auxiliary information for targeted image synthesis.

The classifier-free approach eliminates the need for an additional classifier network, as utilized by Dhariwal et al.[3] by simultaneously training a conditional and an unconditional diffusion model. This approach, introduced by Ho et al., involves selectively dropping the conditioning information at random iterations during the training process [4]. The models are then fine-tuned during sampling by adjusting a guidance scale, which interpolates between the conditional and unconditional noise estimates. This strategy allows the model to find a balance between sample quality and diversity, similar to the effects achieved with classifier guidance [4]. This method poses a significant advantage as it streamlines the training pipeline of the model, essentially eliminating the need to train and maintain a separate classifier model, which would need to be trained on noisy data [4]. By not relying on a separate classifier network, Classifier-Free Guidance significantly simplifies and mitigates the potential adversarial nature of the training process, focusing purely on the model's generative capabilities.

Ho and Salimans present results showing a trade-off between the Inception Score (IS) and the Fréchet Inception Distance (FID) for different guidance scales and probabilities of information omission, achieving a balance that rivals classifier guided performance [4]. Their results with Classifier-Free Guidance show that purely generative diffusion models can produce high-fidelity samples without the need for a classifier, contributing significantly to the field of generative modelling.

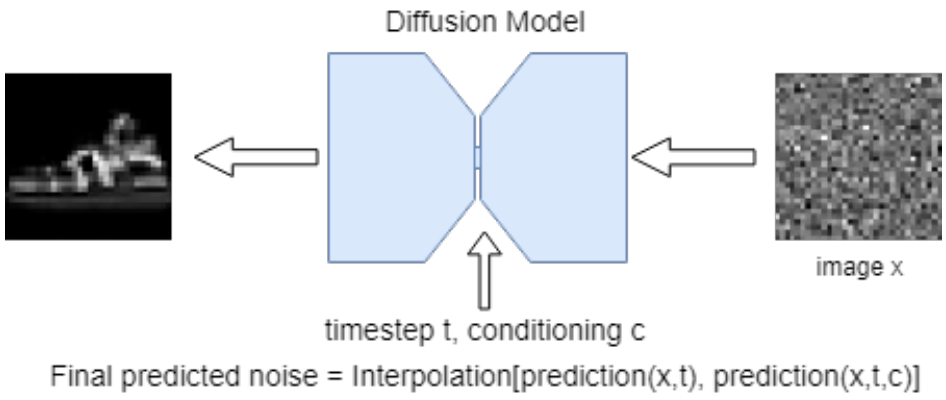


Figure 6: Noise prediction in Classifier-Free Guidance

2.3.2 Conditional GAN (cGAN)

In the technical advancement of Generative Adversarial Networks (GANs), Mirza and Osindero [12] introduced Conditional Generative Adversarial Nets (cGANs) which facilitate guidance within the generative process by conditioning on auxiliary information, such as class labels. This innovation enables the GAN architecture to generate data that is not only realistic but also contextually relevant. The conditional model incorporates information by passing it into both the generator and discriminator networks as an additional input, effectively creating a joint representation of between the data and conditioning information. In essence, this method directs our generator towards specific aspects of data, enhancing the model's utility for tasks that require targeted data generation [12].

Mirza and Osindero demonstrate the utility of cGANs through the generation of MNIST digits conditioned on class labels, highlighting the model's ability to adhere to the conditional constraints. The conditioning in cGANs is implemented through embeddings that represent the conditioning information, enabling the generation of data specific to a given context[12]. This approach significantly contributes to the field by illustrating how generative models can be manipulated to produce high-fidelity outputs that are not just diverse but also aligned with additional input parameters, such as class labels facilitating targeted generation. Their work was instrumental in the development of the conditional DCGAN models within our research.

2.4 Comparative Analysis

The paper "Diffusion Models Beat GANs on Image Synthesis" by Prafulla Dhariwal and Alex Nichol [3] provides a comparative analysis between the performance of diffusion models against Generative Adversarial Networks (GANs) in tasks of image synthesis. Key to the success of the diffusion models presented by Dhariwal and Nichol is the implementation of classifier guidance, which is a method for enhancing sample quality by utilizing gradients from a classifier network for targeted image generation and to direct the generative process towards higher sample fidelity while maintaining diversity [3]. Our comparative analysis differs from their work, as we utilize the concept of Classifier-Free Guidance [4] in our conditional diffusion model.

Dhariwal and Nichol's diffusion models are shown to surpass the state-of-the-art Generative Adversarial Network models, achieving better Inception Scores and Fréchet Inception Distances (FID). Specifically, they attain an FID of 2.97 on ImageNet 128×128, 4.59 on ImageNet 256×256, and 7.72 on ImageNet 512×512 [3]. This achievement was made possible by their novel architecture, cast computational resources, and a deliberate trade-off between diversity and fidelity, achievable through classifier guidance. Notably, the models presented by the authors are competitive even when limited to as few as 25 forward passes per sample, which underscores their efficiency and computational practicality [3].

Their paper also discusses the interplay between classifier guidance and upsampling diffusion models. The authors find that introducing upsampling techniques on the diffusion model generations yields further improvements in FID, with scores reaching 3.94 on ImageNet 256×256 and 3.85 on ImageNet 512×512 [3]. Their architecture and analyses represent significant advancements in the field of generative modeling and demonstrate the potential of diffusion models to create high-quality, diverse images.

The technical underpinnings and the rigorous comparative analysis provided by Dhariwal et al. make their paper [3] a cornerstone reference for our research on the comparative analysis

of generative models. Their findings highlight the viability of diffusion models as a competitive alternative to GANs.

While Dhariwal et al.'s [3] rigorous benchmarks were performed under ideal conditions with multiple machines each equipped with eight high performance NVIDIA Tesla V100 GPUs, our study contrasts this by relying solely on a single accessible GPU. This difference in computational resources underscores the relevance and adaptability of our findings to typical environments encountered by many researchers and developers in the field suffering from a lack of computational resources.

2.5 Evaluation Metrics

The Fréchet Inception Distance (FID) is a metric for assessing the quality of images produced by generative models, specifically focused on comparing the statistical distribution of generated images to that of real images [5]. Introduced in the paper "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium" by Heusel et al.[5], the FID metric has been instrumental in providing a quantitative evaluation that is shown to correlate well with human visual assessment [5]. Their paper shows that the FID operates by extracting feature vectors from an intermediate layer of the Inception-v3 network, a deep Convolutional Neural Network (CNN) pre-trained on the ImageNet dataset. These feature vectors serve as a representation of the images, capturing essential visual information without the final classification layer's constraints. Once feature vectors are extracted for both real and generated images, each set of features is modeled as a multivariate Gaussian distribution with its own mean and covariance matrix.[5]

The mean of the distribution captures the central tendency of the features, meaning a collective measure of where the majority of the data points lie in the feature space. Meanwhile, the covariance matrix encodes the spread and relationship of the feature vectors, effectively denoting the diversity of the images. Therefore, if the generated images have a wide variety of features similar to those in the real dataset, their covariance matrix will more closely resemble that of the real images.[5]

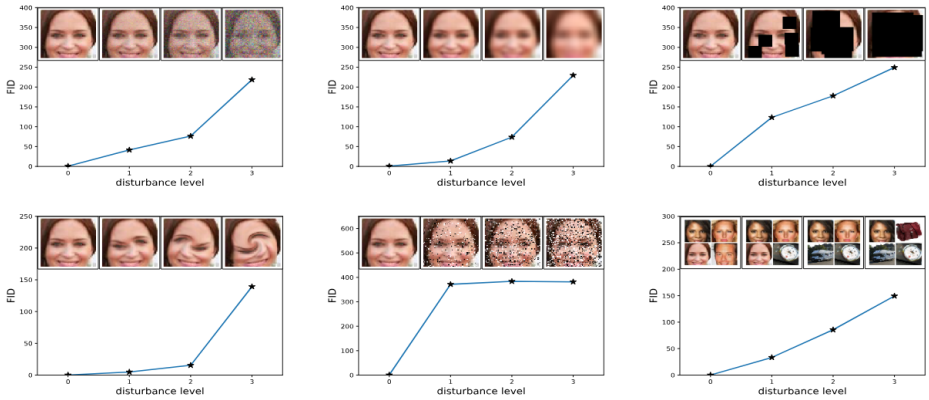


Figure 7: FID evaluated for **upper left**: Gaussian noise, **upper middle**: Gaussian blur, **upper right**: implanted black rectangles, **lower left**: swirled images, **lower middle**: salt and pepper noise, and **lower right**: CelebA dataset contaminated by ImageNet images. (Figure 3 from [5])

The FID score is then calculated using the Wasserstein-2 (also known as Fréchet) distance between the two Gaussian distributions. The Fréchet distance considers both the mean and covariance of the distributions, which makes it sensitive to two critical aspects of image quality: the fidelity of individual images compared to the real data (captured by the means) and the diversity of the images as a whole (captured by the covariances). A lower FID score indicates that the generated images are both diverse and similar to real images, indicating a generative model with better performance.[5]

FID is preferred over other metrics, such as the Inception Score (IS), because it has been found to be more consistent with human judgments of image quality and diversity [5]. This alignment with human perception is what makes FID particularly useful for evaluating generative models. Therefore, we have selected FID as our primary quantitative metric for the comparative analysis of our models.

In essence, FID provides a robust framework for the critical assessment of a model's generative capabilities, enabling researchers to compare different models' performance using a metric that captures the essence of human visual interpretation in a quantitative manner [5].

Note that in Figure 7, salt and pepper noise, black rectangles, and Gaussian noise have a considerable effect on the FID scores of images. This observation will be relevant later, during the evaluation of our models' results (Section 6), when we assess their performance and image generation capabilities.

2.6 Literature Review Summary

The literature review has detailed the theory of Generative Adversarial Networks (GANs) and diffusion models, and discussed their respective methodological and theoretical improvements in subsequent research. GANs, since their conception by Goodfellow et al.[1], have undergone significant refinements such as DCGANs [7] and cGANs [12], each iteration aiming to enhance stability and output quality. Diffusion models have established themselves as robust alternatives, particularly with improvements in sampling efficiency and log-likelihood optimization as presented by Nichol et al. [11]. A blend of these developments has been measured against benchmarks like the Fréchet Inception Distance (FID) [3], which quantifies the quality of generated images in alignment with human visual judgment. Collectively, these findings lay the technical and conceptual foundation, providing the necessary context to appreciate the research and development into generative modeling and the following discussions on the Design (Section 4) and Implementation (Section 5) of the models within our research.

3 Methodology

3.1 Dataset Selection Rationale

We selected the FashionMNIST dataset [6] for its alignment with our project's requirements and constraints. This dataset comprises of 70,000 fashion product images, with 60,000 images in the training set and 10,000 in the test set. Each image has resolution of 28x28 pixels and is in greyscale. Its simplicity permits fast development and training cycles, which was critical given our project's compressed timeframe (discussed further in Project Management Section 8) and the computational resources available. Our project supervisor recommended FashionMNIST [6], recognizing its benchmark status in machine learning and ensuring our results could be contextualized within the broader research landscape.

Nonetheless, using FashionMNIST [6] introduces certain limitations and constrains the overall scope of our research. Since the dataset contains only greyscale images, we do not assess our models' performance on RGB/colored images, which are a more common representation in real-world scenarios. Furthermore, the dataset's images lack the complexity and variation found in higher-resolution datasets. This limitation means our models are not tested against the full spectrum of challenges encountered by generative models when training in a more diverse setting.

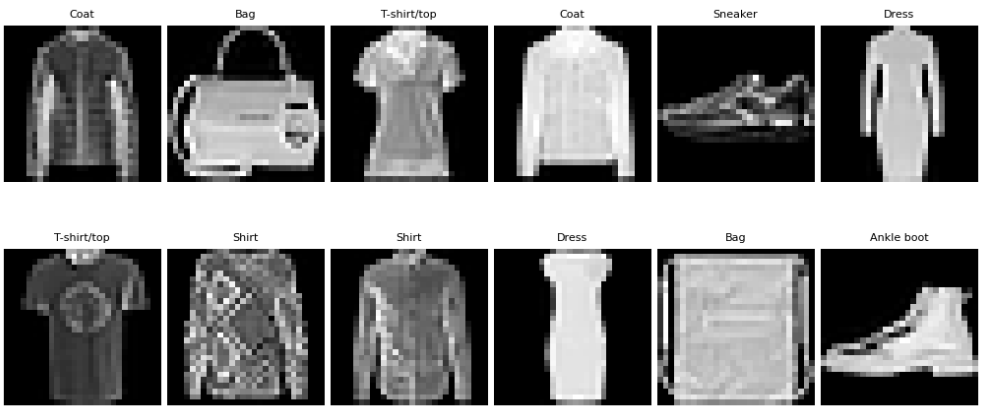


Figure 8: Examples from the FashionMNIST dataset

Despite these constraints, we chose FashionMNIST [6] to strike a deliberate balance between manageable model training and the pursuit of significant research findings. We aimed to work within clear boundaries while still providing insightful contributions to the understanding of generative models' performance, especially in a resource-limited context.

3.2 Model Development Process

We initiated our development process by building unconditional models, which served as a foundational stepping stone towards our class-conditioned systems. These initial models were critical for establishing a baseline understanding of the generative algorithms and for ensuring the correctness of the underlying architecture without the additional complexity of conditioning on auxiliary information.

We draw on the theoretical foundations explored in our literature review (Section 2) and applied key principles to develop these base models. We implemented an iterative approach, starting with simple prototypes of minimal complexity and progressively integrating more sophisticated elements as we validated the functionality of our models. This incremental process allowed us to maintain clarity and manage complexity effectively during the development process.

Once the unconditional models demonstrated stability and a foundational level of performance, we began integrating class label embeddings. The transition to conditional models involved modifying the architecture to accept and process auxiliary categorical data. We carefully planned this phase to ensure that our models not only maintained their previous capabilities but also gained the conditioning functionality without introducing bugs into our code. This development approach aligned well with our Agile development approach which is outlined in Project Management (Section 8).

This step was not without its challenges, as we had to ensure that the introduction of label embeddings did not destabilize the training process of our models. We refined our approach, using feedback from each training session to fine-tune our models, continuously validating

whether the generated images were plausible and also consistent with the provided class labels.

3.3 Evaluation Metrics

We chose Fréchet Inception Distance (FID) scores as the primary metric for quantitatively assessing our models' performance because of the need for a robust and widely recognized measure of image quality. FID scores provide a quantitative measure of similarity between generated images and real images, reflecting both the sample quality and diversity of the generative models' outputs. For further information on the theory/application of Fréchet Inception Distance (FID) scores, please refer to Section 2.5.

4 Design

In this chapter, we detail the architecture and design of the core models for image synthesis used in our research: the unconditional and conditional diffusion models, along with the unconditional and conditional Deep Convolutional Generative Adversarial Networks (DCGANs). This discussion provides insights into the architectural decisions, and the solutions to the challenges encountered during the design of the models.

4.1 Unconditional Diffusion

4.1.1 Variance Scheduling

The idea of variance scheduling in diffusion models is critical for balancing the transition between a noisy distribution and a coherent image output. The scheduler systematically manages the addition of noise through predetermined formulas across the diffusion process, crucial for stability and quality in image synthesis. We initially implement a linear progression of variance, inspired by Ho et al.'s approach in "Denoising Diffusion Probabilistic Models" [8]. The linear schedule is defined as:

$$\beta(t) = \beta_1 + \frac{t(\beta_2 - \beta_1)}{T - 1} \tag{1}$$

where β_1 and β_2 are bounds on the noise levels, T is the total number of diffusion steps, and t is the current timestep. After some tuning, we decided to adopt the parameter values proposed by Ho et al.[8], setting $\beta_1 = 0.0001$, $\beta_2 = 0.02$, and $T = 1000$ for our linear variance scheduler.

Motivated by the improvements detailed in "Improved Denoising Diffusion Probabilistic Models" by Nichol et al.[11], we implemented a cosine variance scheduler. We define the cosine schedule as:

$$\beta_t = \min\left(1 - \frac{\cos\left(\left(\frac{t}{T} + s\right) \div (1+s) \times \frac{\pi}{2}\right)^2}{\cos\left(\left(\frac{t-1}{T} + s\right) \div (1+s) \times \frac{\pi}{2}\right)^2}, \beta_{\max}\right) \tag{2}$$

In this formula, β_t is capped at $\beta_{\max}$ to ensure that the noise level does not exceed a predefined threshold. The variable s acts as a smoothing constant, influencing the starting point of the cosine function to ensure a less abrupt beginning to the diffusion process. The total number of timesteps is denoted by T and t represents the current timestep. This function is designed to be less aggressive in the early stages of diffusion, providing a smoother transition to noised images in the later stages. [11]

4.1.2 Diffusion Process

The forward diffusion process in the unconditional DDPM is designed to systematically introduce noise into clean images, iteratively transforming them into a noised state [8]. This gradual progression is key to the model's capability to generate new images from a noisy distribution, ultimately enabling the generation of high-fidelity images.

The diffusion process incorporates a controlled noise schedule generated by a variance scheduler, to introduce noise at each timestep of the forward process. By precisely adjusting the noise levels, the model ensures a stable transformation of the data, vital for both the quality of the generated images and the efficiency of the reverse diffusion process.

It takes a clean image as input and, depending on the specified timestep, applies a calculated amount of noise [8]. This noise is blended with the original image data using the noising formula:

$$x_t = \sqrt{\alpha_{\text{bars}}(t)} \cdot x_0 + \sqrt{1 - \alpha_{\text{bars}}(t)} \cdot \eta \tag{3}$$

where x_0 is the clean image, $\alpha_{\text{bars}}(t)$ are pre-computed values from the scheduler that dictate the proportion of the original image to retain, and η is the noise added.

4.1.3 U-Net Architecture

The U-Net architecture plays a vital role in the reverse diffusion process of our DDPM, where it predicts the noise added at a specific diffusion step to progressively restore the image. Our

architecture is deeply inspired by a pre-existing U-Net implementation for an unconditional DDPM on GitHub [13]. Our architecture integrates certain modifications to better suit our needs, particularly in handling conditional predictions in future iterations of our DDPM.

Positional Information and Timestep Conditioning Positional information is encoded using sinusoidal embeddings, as suggested by Ho et al.[8], which oscillate between sine and cosine functions at different frequencies. This approach is crucial for maintaining temporal context throughout the diffusion process. Additionally, a small Multi-layer Perceptron (MLP) maps these sinusoidal embeddings to condition the network on the timestep. This conditioning allows for precise adjustments to the network's behavior at each diffusion step, integrating this information into every layer of the U-Net.

Convolutional Blocks Each Residual Convolutional Block within the U-Net is designed to be modular, allowing for easy adjustments and scalability. Every residual block is made up of a series of smaller convolutional blocks that increase the feature space during the downsampling steps and decrease the feature space during the upsampling steps. The smaller convolutional blocks employ Layer Normalization and SiLU (Sigmoid Linear Unit) activation functions strategically to standardize inputs and introduce non-linearities, thus enhancing the learning capabilities of the model.

Downsampling The process begins with several downsampling stages, where the spatial dimensions are systematically reduced while the depth of feature channels is increased. This expansion captures complex patterns essential for the reconstruction phase. The architecture reaches a bottleneck where the most abstract features are processed at the smallest resolution and highest feature depth [9], which is pivotal for the reverse diffusion process.

Upsampling Following the bottleneck, the network undergoes an upsampling process. The spatial dimensions are gradually increased by the residual blocks, and the feature depth is reduced through fractional strided convolutions. We include skip connections from the corresponding downsampling layers to help restore details and context lost during downsampling.

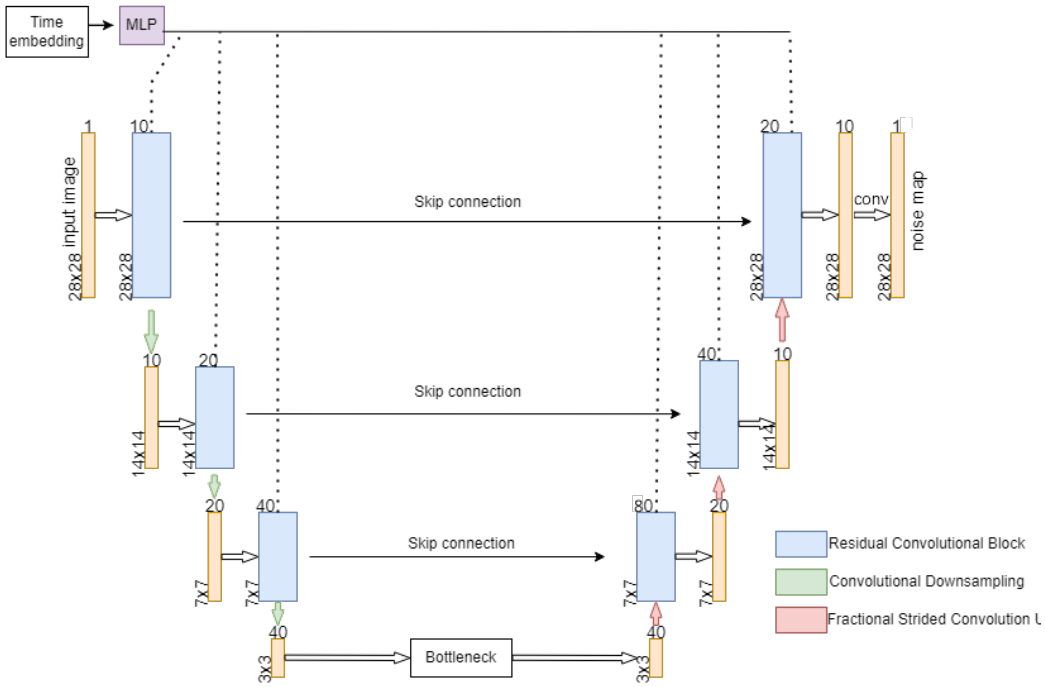


Figure 9: Our U-Net Architecture: This figure depicts the overall layout of our U-Net model. Detailed descriptions of the operations within the Residual Convolutional Blocks are provided in the text. These details have been omitted from the figure to maintain simplicity.

Final Convolution and Denoising The final convolutional layer adjusts the last feature map to predict the noise added at the current timestep t. This prediction is crucial for applying the denoising formula in Equation 4 to compute the image at the previous timestep by removing the predicted noise, effectively recovering the image closer to its pre-noised state.

$$x = \frac{1}{\sqrt{\alpha_t}} \left(x - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_{t-1}}} \cdot \text{predicted noise} \right) + \sqrt{\beta_t} \cdot \text{noise} \tag{4}$$

To improve our understanding and maintainability of the code, we restructured the layout and slightly tweaked the existing implementation [13], introducing clearer segmentation between different components of the network. These modifications were targeted to improve the readability and maintainability of the codebase, making it easier for our future modifications and understanding. We retain the functionality of the original U-Net design from the GitHub implementation [13] while improving readability to allow for future modifications in our conditional diffusion model.

4.1.4 Training Loop

The training loop is a fundamental component of our diffusion model, crucial for teaching the model to predict and subtract noise effectively at each timestep, facilitating the generation of novel images. This section details the structure of our training loop, to ensure efficient learning and optimal model performance.

We first generate noisy images from clean samples by introducing the pre-calculated noise, from our variance scheduler, into the images. Then we sample random timesteps and compute the noised images for those timesteps. Subsequently, the network attempts to reverse this process in its backward pass by predicting the noise added during the specified diffusion step t. By incorporating this dual operation of adding and predicting noise, we enable our model to learn the dynamics of the diffusion process at all the timesteps effectively.

The optimization of the model parameters is conducted through backpropagation, driven by the Mean Squared Error (MSE) loss between the actual noise used to corrupt the images and the noise predicted by our U-Net. MSE loss was chosen due to its simplicity. Our objective is to minimize this error, enhancing the model's ability to accurately predict and subtract noise across various stages of the diffusion process. Future work might look at implementing more complex loss functions as done in the DDPM paper [8] and the Improved DDPM paper [11].

After the processing of each batch in the epoch, the accumulated loss for the epoch is calculated, stored, and displayed. If our model achieves a reduction in the best loss, indicating improved performance, we save the model weights. This approach ensures that the model's best-performing state is preserved, allowing continued training from the most effective parameter set.

Optionally, at the end of each epoch, we incorporate a sampling procedure to visually assess the model's capability of generating images from noise by sampling and displaying images

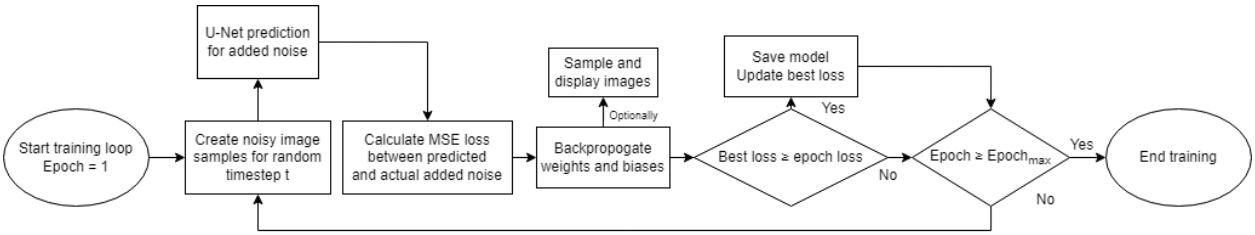


Figure 10: DDPM training loop

from inputs of random noise. This visualization works as an immediate feedback mechanism, providing us with intuitive insights into the model's performance as well as its progress in learning the reverse diffusion process.
This entire training process is presented in Figure 10.

4.1.5 Sampling

The sampling process of our unconditional diffusion model involves starting with samples of noise and reversing the diffusion process to generate images that ideally resemble the original dataset.

We start by creating an initial set of images that are purely composed of noise. We then run these images through the reverse diffusion process, where we set our model to the evaluation mode and predict the noise added at each timestep of the diffusion process for that image. Finally, we utilize the reverse diffusion formula from Equation 4 which incorporates the predicted noise and noise adjustments based on the alpha and beta values, and denoise our images accordingly. Repeating this process for each timestep, we denoise our noisy images until we reach the first timestep, ideally obtaining coherent and novel images that resemble our training data.

4.2 Conditional Diffusion with Classifier-Free Guidance

Our Conditional Diffusion Model extends the capabilities of our Unconditional Model by incorporating class label embeddings to enable targeted image generation. This enhancement is grounded in the principles of Classifier-Free Guidance [4], as outlined by Jonathan Ho

et al., and incorporates strategic insights from "Improved Denoising Diffusion Probabilistic Models" [11] by Nichol et al. This section focuses specifically on the modifications made to the existing Unconditional Model to adapt it for conditional generation, utilizing Classifier-Free Guidance techniques, and omits details on the unchanged code from the unconditional diffusion model.

For details on the implementation of the components of the conditional DDPM that remain unchanged from the unconditional model, readers should refer to the previous section on the unconditional model. Additionally, a comprehensive explanation of Classifier-Free Guidance, including its theoretical underpinnings and applications, is provided in the Literature Review (Section 2). Readers seeking an in-depth understanding of Classifier-Free Guidance are encouraged to consult that section as well as the relevant research paper on the topic [4].

4.2.1 Modified U-Net Architecture

In the U-Net architecture, we have adapted the forward function to condition on both the timestep and class labels as suggested in the Improved DDPM paper, "To make our models class-conditional, we inject class information through the same pathway as the timestep t. In particular, we add a class embedding to the timestep embedding, and pass this embedding to residual blocks throughout the model" [11]. This implementation ensures that every layer of the U-Net is aware of the class context, facilitating more accurate and relevant image generation.

This integration allows the network to use class information to refine its predictions for each timestep, effectively tailoring the reverse diffusion process to generate images that not only reconstruct the original image but also align closely with the learned attributes/features of the specified class.

The diagram depicting the architecture of this U-Net is omitted as the modifications from the previously shown architecture are minimal. Essentially, the architecture remains the same as illustrated in Figure 9. However, instead of passing only the sinusoidal time embedding

as the input to the MLP, we now concatenate this with a reshaped class label embedding before feeding it into the MLP and passing this new context embedding which contains both temporal and class information to the residual blocks of our U-Net.

4.2.2 Modified Training Loop

The training loop has been modified to drop the conditioning on labels for a certain percentage of the iterations, a technique that is part of the Classifier-Free Guidance strategy [4]. This introduces a regularization effect, helping the model generalize better and avoid overfitting to specific class features. By randomly omitting the class label information during training, the model learns to not only rely on class labels but also preserve its ability to generate unconditional images based on the diffusion process alone [4]. In essence, we are training an unconditional and a conditional model at the same time.

This sets us up to generate unconditional and conditional predictions during the sampling process, a critical aspect of Classifier-Free Guidance [4].

4.2.3 Modified Sampling Procedure

The sampling process in the conditional DDPM must be adjusted to accommodate for class labels. When generating new images, the model should use the class labels to steer the generation process, employing a guidance scale as suggested in the Classifier-Free Guidance paper [4]. The model should interpolate between conditioned and unconditioned noise predictions using the guidance scale to denoise the image.
The Conditional Diffusion Model represents a significant advancement in our ability to generate targeted, high-quality images through a guided diffusion process. By integrating class labels directly into the reverse diffusion steps, we harness the full potential of Classifier-Free Guidance [4]. These enhancements ensure that the model not only generates images that are visually appealing but also closely aligned with specific class characteristics.

4.3 Unconditional Deep Convolutional GAN

The unconditional Deep Convolutional Generative Adversarial Network (DCGAN) represents a cornerstone of our generative modeling efforts, aiming to produce high-quality images without the use of labeled training data. This implementation is inspired by and adapted from a well-documented Keras implementation provided by Jason Brownlee [14], which has been reimplemented in PyTorch, leveraging specific suggestions detailed within the key GAN publications [1, 7, 12].

4.3.1 Overview of DCGAN Architecture

The DCGAN architecture utilizes CNNs for both the generator and discriminator, enhancing the stability of training over standard fully connected GANs [7]. This architecture has been proven effective in generating coherent and visually appealing images [7]. Our implementation adheres to the principles laid out in the foundational GAN literature [1, 7], ensuring robust training dynamics and efficient learning of data distributions.

4.3.2 Discriminator Architecture

The discriminator acts as a binary classifier, attempting to distinguish between real images from the dataset and fake images produced by the generator [1].

The discriminator consists of convolutional layers that progressively downsample the input

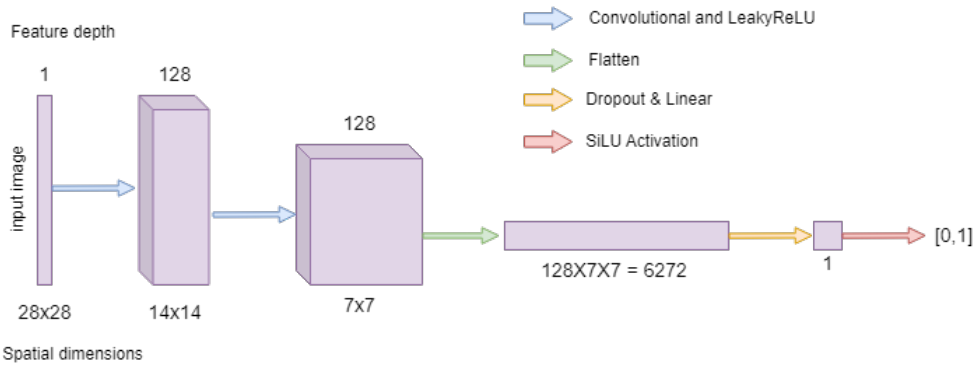


Figure 11: Our DCGAN Discriminator Architecture. Note that normalization layers have been omitted for brevity.

image, coupled with LeakyReLU activations to maintain gradient flow during training. The use of Dropout regularization aims to reduce overfitting and improve model generalization. By flattening the output of the convolutional layers and passing it through a SiLU activation, the network outputs a probability indicating the likelihood of the input being a real image.[7]

The architecture of our discriminator network can be visualized in Figure 11.

4.3.3 Generator Architecture

Conversely, the generator synthesizes fake images from latent noise vectors. Its structure is crucial for accurately producing high-quality images.

The generator takes a high-dimensional latent vector of noise as input and applies a linear layer that projects and reshapes the latent vector into a small spatial dimension, which is then upscaled through fractional strided convolutional layers[7]. Each upsampling step is normalized and activated using LeakyReLU to ensure effective training and stabilization.The final convolutional layer uses a Tanh activation to output the image, which normalizes pixel values between -1 and 1, matching the pre-processing for the training images.[7]

In line with the foundational DCGAN paper [7], which specifies a "100-dimensional uniform

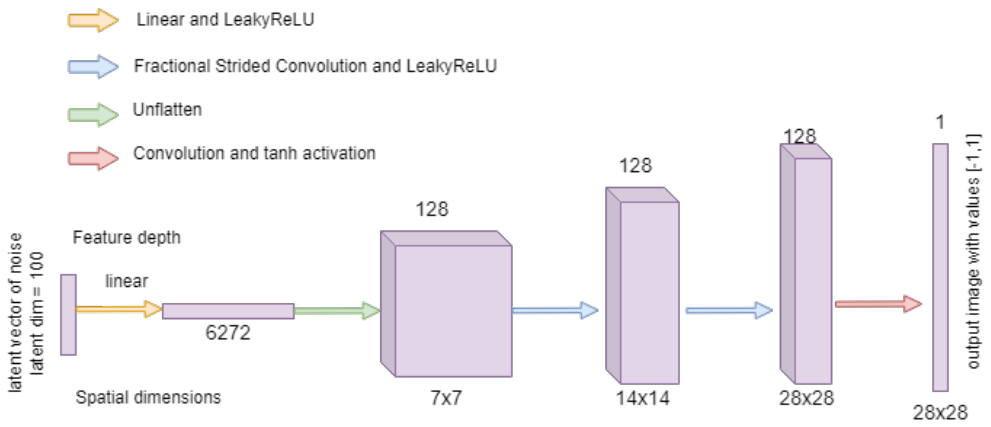


Figure 12: Our DCGAN Generator Architecture. Note that normalization layers have been omitted for brevity.

distribution Z is projected to a small spatial extent convolutional representation with many feature maps" within their convolutional generator, our implementation also adopts a latent dimension of 100.

The architecture of both the discriminator and generator networks forms the base of our unconditional DCGAN, establishing a solid foundation for synthesizing images. This framework also facilitates the integration of conditional elements in the following conditional DCGAN model. Drawing on insights from related research [1, 7] and the Keras implementation [14], we have developed a stable architecture for our generator, which is depicted in Figure 12.

4.3.4 Training

The training loop for the unconditional DCGAN is structured to optimize both the generator and discriminator networks through adversarial training. We setup this process to alternate between training the discriminator to accurately classify images and training the generator to fool the discriminator.

The discriminator is trained first in each cycle by feeding it real images from the dataset and fake images produced by the generator. The goal is to maximize its accuracy in distinguishing between synthetic and authentic.

Subsequently, the generator is trained to produce images that the discriminator classifies as real. This involves updating the generator's weights based on how well it manages to trick the discriminator.

4.3.5 Loss and optimizations

We utilize Binary Cross-Entropy (BCE) loss to measure the performance of the generator and discriminator as suggested in the Keras implementation [14]. In an ideal scenario, the discriminator would converge to a point where it assigns a probability of 0.5 to both real and synthetic images. This would indicate that it can no longer distinguish between real and fake, corresponding to a BCE loss value of approximately 0.693. This represents the

natural logarithm of two ln(2), the expected loss when the predicted probability equals the equilibrium probability of 0.5. However, during training, the networks often experience oscillations in the loss values due to the adversarial nature of the training procedure, where the generator and discriminator continuously adapt and respond to each other's "strategies".

We utilize separate Adam optimizers for the generator and discriminator and tune the optimizers for both based on insights derived from the DCGAN study. Specifically, the DCGAN authors noted, "We found the suggested learning rate of 0.001, to be too high, using 0.0002 instead" [7]. Following this guidance, we also used the reduced learning rate of 0.0002. Additionally, they observed that "leaving the momentum term β_1 at suggested value of 0.9 resulted in training oscillation and instability while reducing it to 0.5 helped stabilize training" [7]. Encouraged by their findings, we similarly adjusted the β_1 value to 0.5 to ensure stability within our training process.

After each epoch, we optionally display a batch of images (as we did in the DDPM) generated from latent noise vectors. This allows us to visually review the generator's progress in creating realistic images as training progresses through the epochs.

The structured adversarial training regime ensures that both networks learn effectively, with the discriminator becoming adept at classifying images with increasing accuracy and the generator enhancing its ability to create indistinguishable synthetic images. This adversarial training loop is crucial in driving results in the DCGAN towards generating high-fidelity images.

4.4 Conditional Deep Convolutional GAN

Building upon the architecture and foundational principles of the unconditional DCGAN, we introduce enhancements to create a conditional Deep Convolutional Generative Adversarial Network (cDCGAN). We achieve this by integrating class label embeddings into both the generator and discriminator as suggested by Mirza et al.[12], enabling the generation of images conditioned on specific class features. These changes enhance the model's capability

to generate diverse and class-specific images, in line with the modifications described in the Keras implementation [14].

4.4.1 Modified Discriminator Architecture

We adapt the discriminator in the conditional DCGAN to take class labels as an additional input. This is achieved by embedding the class labels into a continuous vector space and reshaping these embeddings to match the spatial dimensions of the input images (1,28,28), essentially forming an additional image-like channel which is then concatenated with the actual image input [12]. We change the input channel size to accommodate the additional class information, effectively treating the concatenated label embedding as part of our input image. This enables the discriminator to use information specific to the class when trying to classify an image as real or synthetic.

The diagram for this architecture is omitted. In essence, we have the same architecture as in Figure 11, but our input image now has a concatenated label embedding along the channel dimension making the data shape (2,28,28), which we then pass through the same convolutions.

4.4.2 Modified Generator Architecture

Similarly, we modify the generator to incorporate class labels into the generation process. This involves embedding the class labels as we did in the modified discriminator, by creating a class label embedding in a continuous vector space and then reshaping it to match the dimensionality of our generator input. The transformed label embeddings are concatenated with the latent vector transformed feature maps, effectively increasing the input channel size of the first upsampling layer in our Generator [12].

The diagram for this architecture is omitted for brevity. In essence, we have the same architecture as in Figure 12. First we create an embedding of our class label in a high-dimensional vector space which we pass through a linear layer to reshape it to (1,7,7). Then we pass our latent noise vector through a linear layer with LeakyReLU and BatchNorm making it

(128,7,7). We then concatenate our latent noise and class label embedding to get data of the shape (129,7,7) which we pass through thefractional strided convolutions as we previously did in the unconditional DCGAN.

These adaptations ensure that both the discriminator and generator of the cDCGAN effectively utilize class information to generate and evaluate images. This allows for targeted image generation specified on class labels, enhancing the model's utility and generating images of higher fidelity as we found in the quantitative evaluation (Section 6.3.2).

4.4.3 Modified Training Loop

We then modify the training loop for the cDCGAN to incorporate class labels into the adversarial training process. Both our discriminator and generator are trained with awareness of the class labels, making the networks learn to recognize class-specific attributes. This enhances their ability to produce and evaluate class-conditional images [12].

These modifications are vital for the effective training of the cDCGAN, allowing it to generate diverse and accurate class-specific images while maintaining its characteristic adversarial dynamic.

5 Implementation

This section covers the practical implementations of the architectural designs discussed in the previous section. It includes details about the development environment and presents the implemented algorithms, supported by performance evaluations.

5.1 Development Environment

Python 3.8 was selected as the programming language for developing the generative models, due to its status as the most dominant language in machine learning. This choice was further reinforced by Python's comprehensive libraries and support within the active machine learning community. This was complemented by CUDA version 12.1, enabling GPU acceleration to train our deep learning models as efficiently as possible. The deep learning framework PyTorch was recommended by the project supervisor due to its extensive documentation and community support. We used Jupyter Notebooks for development, as it supports live code and visualization features which proved helpful for fast debugging.

The model training was carried out on an NVIDIA RTX 3060 GPU. This setup facilitated rapid iteration cycles which proved crucial for the Agile methodology that was later adopted in this project. For further information on this please refer to Section 8.

5.1.1 Code Availability

All source code developed for this project, including scripts for model training, data processing, and quantitative metrics calculations, is available on GitHub.
The code can be accessed at [`https://github.com/samirrajes/dissertation-project`].

5.2 Unconditional Diffusion

5.2.1 Variance Scheduling

In the implementation of our unconditional diffusion model, we carefully designed the variance scheduler to modulate the noise addition effectively across the diffusion process. The

implementation allows us to choose between linear and cosine noise schedules based on the specified scheduling type.

The implementation of the variance scheduler (Algorithm 1) facilitates the computation of the diffusion parameters needed for varying the noise level throughout the model's forward process in a controller manner. By establishing a variance schedule based on either linear or cosine methods, the model can adapt the noise introduction effectively, allowing for controlled noising and subsequent reconstruction of image data. The strategic manipulation of noise levels is essential for training the diffusion model to reverse the noise addition accurately and generate high-fidelity images.

We plot the difference between the linear and cosine schedule in Figure 13, where $\alpha_{\bar{t}}$ represents the residual clarity of the image at the timestep t. This plot aligns with the plot of the cosine and linear schedulers in the paper by Nichol et al.[11], confirming that both our schedulers function as intended. We incorporate the recommended values from Nichol et al.'s paper [11], with $\beta_1 = 0.0001$, $\beta_2 = 0.02$, $T = 1000$, $s = 0.008$, and $\beta_{\max} = 0.999$.

Algorithm 1 Variance Scheduler Implementation

Input: β_1, β_2, T, *schedule_type* = "linear", $s = 0.008$, $\beta_{\max} = 0.999$
Output: Computed noise schedule with $\beta(t)$, $\alpha(t)$, and $\alpha_{\text{bars}}(t)$
Initialize timesteps from 0 to $T - 1$
if *schedule_type* == 'linear' **then**
    Compute linear betas for each timestep:
    Computed using Equation 1
else if *schedule_type* == 'cosine' **then**
    Compute betas using cosine schedule:
    Computed using Equation 2
else
    Raise error: "Unknown schedule type"
end if
Compute $\alpha(t) \leftarrow 1 - \beta(t)$
Compute cumulative product $\alpha_{\text{bars}}(t)$
return {'betas': β, 'alphas': α, 'alpha_bars': α_{bars}}

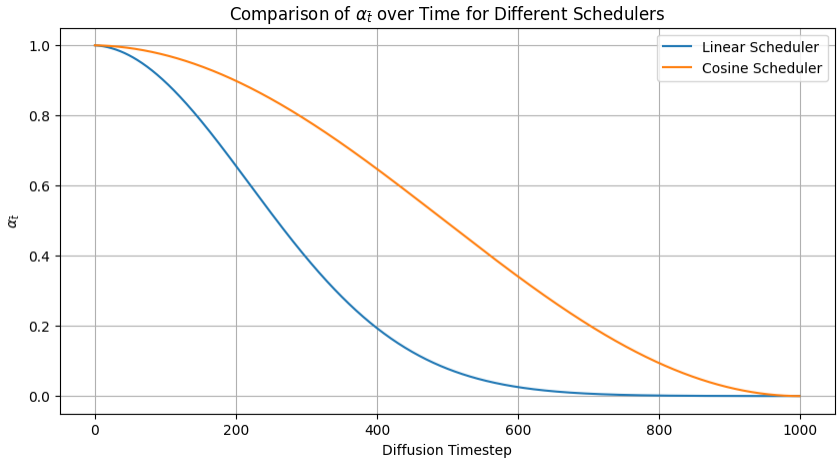


Figure 13: $\alpha_{\bar{t}}$ plotted against diffusion timestep

5.2.2 Forward Diffusion

The implementation of the forward diffusion process (Algorithm 2) involves applying the noising equation (Equation 3) detailed in our design (Section 4.1.2) to apply controlled noise from our scheduler to an image based on a timestep t.

The process begins by retrieving the predefined noise schedule from the variance scheduler, specifically α_{bars} for the given timestep t. This is crucial as it determines the level of noise to be added to the clean image x_0. If the noise η is not explicitly provided, the function generates random Gaussian noise, which matches the dimensions of the input image.

We then apply the noise to the clean image. This is achieved by a weighted combination where $\sqrt{\alpha_{\text{bars}}(t)} \cdot x_0$ represents the proportion of the original image retained, while $\sqrt{1 - \alpha_{\text{bars}}(t)} \cdot \eta$ represents the noise added. The resulting image x_t, now a noised version of the original, encapsulates the effect of the noise at that particular timestep.

This represents the core logic in the forward function of the DDPM model, describing how the model calculates the noisy image at any given timestep using the pre-computed variance schedule. The process allows direct computation of the noised state, bypassing incremental steps if needed, and uses either provided or generated noise depending on input. We test

Algorithm 2 Diffusion function

Input: Clean image x_0, timestep t, optional noise η
Output: Noised image x_t
Retrieve α_{bars} for timestep t from the variance scheduler
if η is not provided **then**
    Generate η as random noise matching the dimensions of x_0
end if
Compute noised image:
$x_t \leftarrow \sqrt{\alpha_{\text{bars}}(t)} \cdot x_0 + \sqrt{1 - \alpha_{\text{bars}}(t)} \cdot \eta$
return x_t

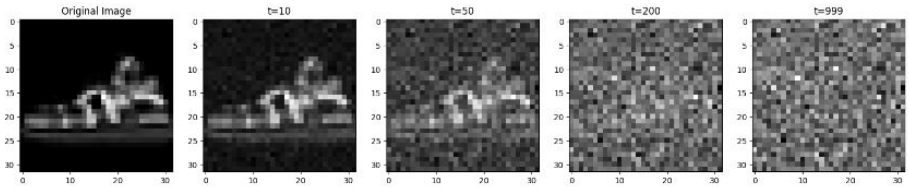


Figure 14: Diffusion Process with Linear Schedule. Note that the timesteps are not linearly spaced in this figure.

our function by plotting the noising process for 1000 timesteps, seen in Figure 14.

5.2.3 Unconditional U-Net

The implementation of the U-Net in our unconditional diffusion model draws heavily from a pre-existing implementation available on GitHub [13]. We obtained explicit permission from the author to adapt this implementation for use in our research. The network is a detailed configuration of layers and operations tailored for the reverse diffusion process. We present a simplified pseudocode that outlines the key components and flow of our U-Net architecture in Algorithm 4.

A smaller convolutional block is implemented in Algorithm 3. These convolutional blocks are applied sequentially, to make up a Residual Convolutional Block. The Residual Convolutional Blocks increase the data's feature space during the downsampling steps, and decrease its feature space in the upsampling steps.

We utilize convolutional layers between the residual blocks in the downsampling stage to

Algorithm 3 Small Convolutional Block

Input: Input tensor x, Input channels in_c, Output channels out_c, Kernel size $kernel\_size = 3$, Stride $stride = 1$, Padding $padding = 1$
Output: Transformed tensor after applying convolutional block
Apply Layer Normalization to x if normalization is enabled
Convolution operation on x with parameters in_c, out_c, $kernel\_size$, $stride$, $padding$
Apply SiLU activation function
Repeat convolution and activation
return Resulting tensor from second activation

Algorithm 4 U-Net forward pass

Input: Image x, Timestep t
Output: Noise map for image x_t at timestep t
Create sinusoidal embedding for current timestep t
Initialize time-dependent embeddings and apply to x
Downsampling:
for each downsampling block **do**
    Apply Residual Block (Made up of Convolutional Blocks from Algorithm 3).
    Reduce spatial dimension, increase feature depth via convolution
end for
Bottleneck:
Process through bottleneck layers with maximal feature compression
Upsampling:
for each upsampling block **do**
    Increase spatial dimension, reduce depth via fractional strided convolution
    Merge features from corresponding downsampling layer (skip connection)
    Apply Residual Convolutional Block
end for
Final Adjustments:
Adjust through final convolutional layer to predict noise at current timestep t
(Compute the noise map added to the image at the specified timestep)
return the noise map

decrease spatial dimensions, and fractional strided convolutions in the upsampling stage to increase spatial dimensions. The use of skip connections is vital to the upsampling stage as it helps the U-Net recover details lost during downsampling.

This architecture is depicted in detail in the design section (Figure 9), providing a visual representation of the intricate flow of data through the U-Net, from input to output, showcasing the critical role of each component in achieving effective image reconstruction.

Algorithm 5 Unconditional DDPM Training Loop

Initialize: best_loss = ∞, losses = []
for epoch = 1 **to** final epoch **do**
    Initialize epoch_loss = 0.0
    for each batch of images **in** the dataset **do**
        Generate random noise η and select timesteps t
        Compute noisy images
        Predict noise η_θ of noisy images
        Compute MSE loss loss between η and η_θ
        Perform backpropagation and optimizer step
        Accumulate the epoch loss
    end for
    Store epoch loss in losses for plotting
    Optionally display generated images
    Log and save model if epoch_loss improves
end for
Plot the training losses over epochs

5.2.4 Training Loop

The training loop for our Unconditional DDPM model is structured to efficiently learn and refine the model's ability to predict and reverse noise in images. This process is crucial for the development of a robust diffusion model capable of high-quality image synthesis.

As shown in the training loop (Algorithm 5) and detailed in the design (Section 4.1.4), we pre-compute noise for images using a variance scheduler after which we sample random timesteps and images at those timesteps. We then compute the predicted noise using our U-Net. After, we calculate the loss between the predicted and actual noise for an image at a timestep using Mean Squared Error (MSE) loss and update our weights and biases accordingly.

Additionally, we store our model when we obtain a new low in the loss for the epoch, and we print the losses over the epochs at the end of our training loop as seen in Figure 15.

Algorithm 6 Sampling Process in Diffusion Model

Ensure model is in evaluation mode.
Generate initial noisy image samples
for t from last timestep down to first timestep **do**
    Condition on current timestep t to get predicted noise.
    Retrieve α_t, α_{t-1}, and β_t for current timestep.
    Calculate the noise adjustment for this step.
    if $t > 1$ **then**
        Generate random noise noise.
    else
        Use no noise for the final adjustment.
    end if
    Update x using the reverse diffusion formula
end for
Return the denoised images.
Set the model back to training mode.

5.2.5 Unconditional Diffusion Sampling

As seen in Algorithm 6, we implement the iterative process by which the model progressively removes noise from noisy images attempting to produce novel images that resemble the data distribution. We initialize the model in evaluation mode to ensure consistent output and begin iteratively denoising the images from the last to the first timestep, applying noise prediction and correction based on the reverse diffusion formula (Equation 4) using α_t, α_{t-1}, and β_t. Our denoising process includes the introduction of stochastic noise for all but the final timestep, where no additional noise is added. The efficacy of the sampling process is evident

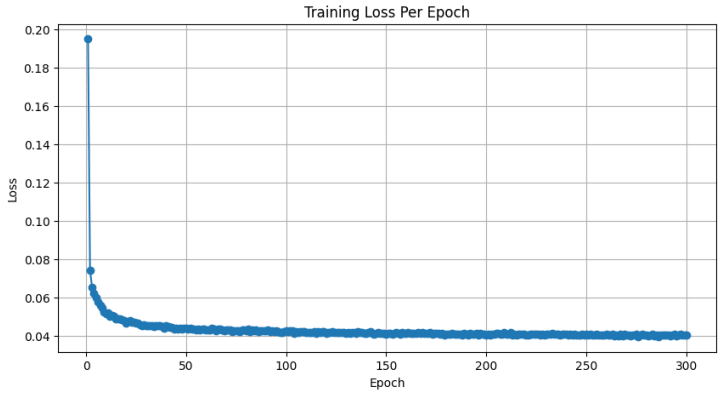


Figure 15: Unconditional DDPM Loss over 300 epochs

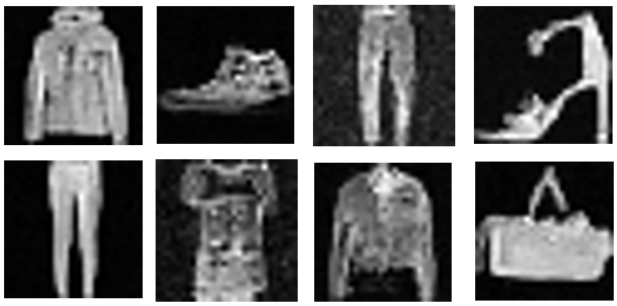


Figure 16: Unconditional DDPM sampled images after training for 100 epochs

by the samples illustrated in Figure 16, demonstrating the model's ability to transform noisy inputs into coherent articles of clothing. However, there is some residual noise in some of the images that we hope to mitigate through training for a larger number of epochs. After sampling is finished, the model reverts to training mode.

5.3 Classifier-Free Guidance Implementation

An advantage of utilizing Classifier-Free Guidance [4] lies in its minimal modification requirements to adapt an unconditional DDPM for class-conditional generation. In the following sections, we detail the modifications applied to our U-Net architecture, training loop, and sampling procedure to transition our model from unconditional to class-conditional.

5.3.1 Conditional U-Net

We need to condition our U-Net on class label information in addition to the timestep information. We achieve this by implementing the suggestion from Nichol et al.'s paper [11] that we detailed in our design section 4.2.1. We create a class label embedding, reshape it, and concatenate it to the time step embedding. We then pass this concatenated embedding through the MLP as we did in the unconditional diffusion model and condition every layer of the U-Net on it. The modified aspects of the U-Net are detailed in Algorithm 7, with the unchanged aspects omitted for brevity.

Now that our U-Net is conditional on class labels, we need to modify the training procedure to pass class labels to the U-Net when making predictions.

Algorithm 7 Modified Forward Function of Conditional U-Net

Input: Image x, Timestep t, Class labels $labels$
Output: Noise map for image x at timestep t
for t from last timestep down to first timestep **do**
    Retrieve timestep embedding for t.
    Retrieve class embedding for $labels$.
    Concatenate the timestep and class embeddings to form the combined embedding.
    Propagate x through the network using the conditioned embedding:
    Rest remains unchanged...
end for
Return the noise map for noise added to image x at timestep t.

5.3.2 Conditional Diffusion Training

The Classifier-Free Guidance paper [4] shows that we need to now pass class labels to our U-Net while training, but also omit the class conditioning information for a certain percentage of training iterations. They experiment with different probabilities in their paper and suggest dropping the conditioning information for 10-20% of the training iterations, to essentially train a conditional and unconditional model at the same time. We utilize a 10% chance of omitting class label conditioning in our implementation, as depicted in Algorithm 8. Only the modified aspects of the training loop are depicted, with unchanged functionality omitted for brevity.

Now that we are training our model to be able to predict noise regardless of if we provide it with a class label or not. We need to modify our sampling procedure to utilize both types of noise predictions in its image generation procedure.

Algorithm 8 Modified Training Loop with Classifier-Free Guidance

for epoch = 1 **to** final epoch **do**
    for each batch $(x, labels)$ **in** dataset **do**
        Generate a random flag with 10% probability to drop labels
        if $t > 1$ and flag indicates to drop labels **then**
            Set $labels$ to None
        end if
        Perform forward pass with or without labels based on mask
        ... Rest remains unchanged
    end for
end for
return Trained Model

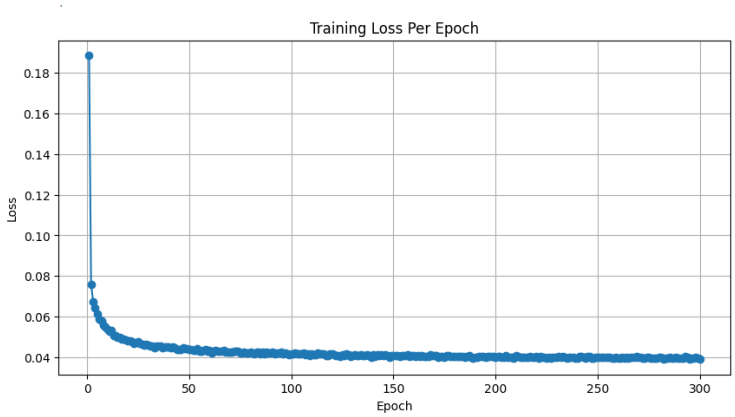


Figure 17: Conditional DDPM Loss over 300 epochs

5.3.3 Conditional Sampling Procedure

We modify our sampling procedure to let us sample an image based on a specific class label. We do this by making 2 predictions for the noise: one conditioned on the class label, and one not conditioned on any auxiliary information (aside from the timestep). We then linearly interpolate between these two predictions using a guidance scale. Different values of the guidance scale were tested in the Classifier-Free Guidance paper [4], showing that the higher the guidance scale the more the image correlated with the specified class. However, for a higher guidance scale FID scores are negatively impacted, probably due to a decrease in diversity of the produced images.

In this dissertation, we opt for a guidance scale of 3.0 to obtain a balance between diversity and quality. Our implementation of the sampling procedure is shown in Algorithm 9. Only the modified aspects of the sampling procedure are depicted, with unchanged functionality omitted for brevity.

Generating and displaying some samples as illustrated in Figure 18, shows that there seems to be improvements in our conditional DDPM after training on class labels for 300 epochs, however upon visual inspection the generated samples seem to lack diversity. We speculate that this is due to the guidance scale being too high. Effective tuning of the guidance scale is left to future work due to time constraints.

Algorithm 9 Enhanced Sampling Process in Conditional Model

```
.. Setup remains unchanged
Initialize a guidance scale
for t from last timestep down to first timestep do
    Generate conditioned prediction
    if guidance scale > 0 then
        Generate unconditioned prediction by omitting label information
        Interpolate between unconditioned and conditioned prediction using scale
        Utilize this interpolated noise for denoising calculations
    end if
    .. Rest remains unchanged
end for
```

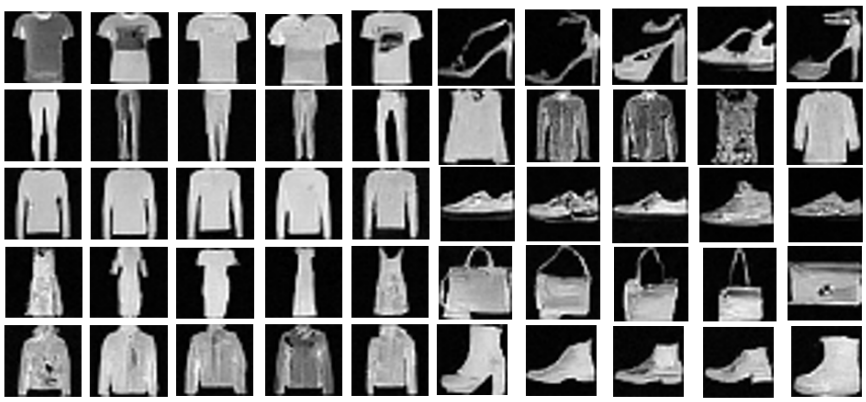


Figure 18: Conditional Diffusion sampled images after training for 300 epochs

5.4 Unconditional Deep Convolutional GAN Implementation

5.4.1 Discriminator

Following the architecture we detailed in the design (Section 4.3.2) of our unconditional discriminator, we utilize convolutional layers to reduce the spatial dimensions of an image

Algorithm 10 Discriminator Model of DCGAN

```
Input: Image x
Output: Classification probability
Perform convolution with 128 filters, kernel size 3, stride 2, padding 1
Apply LeakyReLU with negative slope 0.2
Perform another convolution with 128 filters, kernel size 3, stride 2, padding 1
Apply LeakyReLU with negative slope 0.2
Flatten the output
Apply dropout with probability 0.4
Dense layer to map flattened output to 1 unit
Apply sigmoid activation to output
return Classification probability
```

until we can flatten the image into a vector representation and apply a linear transformation alongside a SiLU activation to generate a probability the image is real/fake.
The specific details and values of our implementation are seen in Algorithm 10.

5.4.2 Generator

The architecture of our generator is detailed in Section 4.3.3. In essence, we sample a noisy latent vector from a latent dimension and reshape it to a representation with a high dimensional feature space and low spatial dimensions (128,7,7). We then apply fractional strided convolutions alongside LeakyReLU activations and BatchNorm to upsample the image. For the final layer we apply a convolutional layer and a tanh activation function to normalize our image outputs to [-1,1]. The specifics of our implementation are detailed in Algorithm 11.

5.4.3 Training

Now we implement the adversarial training process inherent to the GAN architecture. The training loop depicted in Algorithm 12 follows a standard GAN training regime where the

Algorithm 11 Generator Model of DCGAN

1: **Input:** Latent vector z
2: **Output:** Generated image
3: Map z through a dense layer to dimension $128 \times 7 \times 7$
4: Apply LeakyReLU with negative slope 0.2
5: Apply Batch Normalization
6: Unflatten output to shape $128, 7, 7$
7: Upsample to 14×14 using fractional strided convolution with 128 filters, kernel size 4, stride 2, padding 1
8: Apply LeakyReLU with negative slope 0.2
9: Apply Batch Normalization
10: Upsample to 28×28 using fractional strided convolution with 128 filters, kernel size 4, stride 2, padding 1
11: Apply LeakyReLU with negative slope 0.2
12: Apply Batch Normalization
13: Produce final image using a convolution with 1 filter, kernel size 7, padding 3
14: Apply Tanh activation to the output
15: **return** Generated image

generator and the discriminator are adversarially trained in a zero-sum game.
We train the discriminator to accurately distinguish between real and synthetic images produced by the generator. During training we feed it real images first, training it to predict that they are real. We then feed it synthetic images produced by our generator training it to identify these as fake. We adjust the discriminator's parameters based on the losses calculates from both the real and synthetic images to drive accurate predictions.

We train the generator with the goal of fooling the discriminator by creating images that convincingly represent the training dataset.
We generate images using our generator which are then tested against the discriminator. The loss received is used to adjust the generators parameter's such that it will produce more realistic images in future iterations.

We employ Binary Cross Entropy (BCE) Loss as a measure of performance for both net-

Algorithm 12 Training Loop for Unconditional GAN

```
Initialize optimizers and loss criterion (Binary Cross Entropy loss)
Initialize lists to track losses
for epoch = 1 to final epoch do
    for each batch (images, _) in dataset do
        Prepare real labels and fake labels
        Train Discriminator:
            Calculate loss on real images
            Generate fake images using generator
            Calculate loss on fake images
            Update discriminator using average loss
        Train Generator:
            Generate new fake images
            Calculate loss using updated discriminator
            Update generator
        Log batch losses and update progress display
    end for
    Calculate and store average losses for epoch
    Optionally generate and display images at the end of the epoch
end for
Save final trained generator model
Display training loss graphs for both models
```

works. This loss quantifies how robust the discriminator is in differentiating between real from fake, and how well the generator can fool the discriminator.

Each network has its own Adam optimizer, which updates weights based on the BCE loss. At the end of each epoch, we calculate the average losses and store them for performance tracking. At the end of the training loop we plot the discriminator's and generator's losses over the training epochs as seen in Figure 19. This provides insights into how their adversarial training dynamics work.

Optionally, as we did with the diffusion models, we can sample and display images generated by the generator at the end of each epoch. This visual feedback allows us to qualitatively assess improvements in image quality and realism.

5.5 Conditional Deep Convolutional GAN Implementation

5.5.1 Conditional Discriminator Modifications

As detailed in the design of our conditional discriminator (Section 4.4.1), and in the Algorithm 13, we take the class label as an input to our discriminator alongside an image, and create a label embedding that we concatenate to the input image along the feature space. In essence, while in our unconditional discriminator we had an initial input size of (1,28,28)

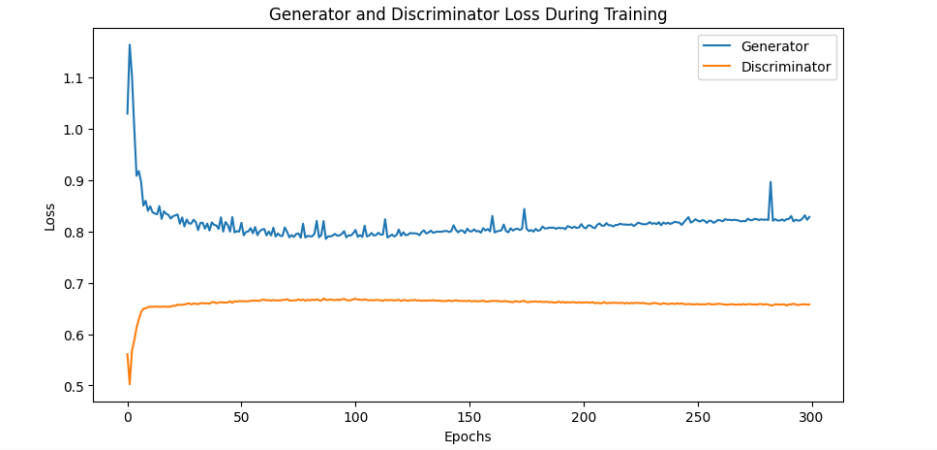


Figure 19: DCGAN loss over 300 epochs

Figure 20: DCGAN samples after 300 epochs of training

Algorithm 13 Conditional Discriminator Model of DCGAN

1: **Input:** Image x, Class labels *labels*
2: **Output:** Classification probability
3: Initialize label embedding to match the image size 28×28
4: Embed labels and reshape to the dimensions of the input images
5: Concatenate embedded labels with images along the channel dimension
6: **Rest of the model operations are unchanged**
7: **return** Classification probability

for our grayscale image, here we reshape the class label embedding to the size of the image, and concatenate them together forming an input shape (2,28,28). This tensor is then passed through the same layers as in our unconditional discriminator to generate a prediction on whether the image is real or fake, however in this implementation it is conditioned on the class label information as well.

5.5.2 Conditional Generator Modifications

Similarly, our conditional generator also works through the use of concatenation albeit with a few extra steps as detailed in Algorithm 14. We reshape our latent vector of noise as we did before to get a high dimensional feature space with a low spatial dimension (128,7,7). Then we create our class label embedding, mapping it to a higher dimensional vector. Then we reshape our embedding to dimensions that support concatenation with the latent vector of noise, in this case the embedding is reshaped to (1,7,7). Concatenating these two together leaves us with a tensor of size (129,7,7) representing our noisy latent image with label information within it. We then run this through the same unconditional GAN architecture to obtain our class-conditioned generated image.

Algorithm 14 Conditional Generator Model of DCGAN

Input: Latent vector z, Class labels *labels*
Output: Generated image
Initialize label embedding to transform label dimensions to feature map dimensions
Embed labels and resize using a linear layer to 7×7 feature map size
Initialize latent vector transformation as before
Concatenate label and latent feature maps:
    Combine transformed labels and latent vectors along the channel dimension
Rest of the model operations are unchanged
return Generated image

5.5.3 Conditional Training loop Modifications

Now we need to update our adversarial training loop to incorporate our conditioning information. As seen in Algorithm 15 there is very little to change from the unconditional training loop. Essentially, we now train the discriminator on real images as before but with their corresponding labels passed to the discriminator as well. Then we train the discriminator on fake images generated based on fake labels fed into the generator. We update the parameters after calculating the BCE Loss the same way we did before. Then we train the generator by generating fake images with fake labels which we pass to the discriminator and calculate the loss. We update the parameters as we did in our unconditional GAN.

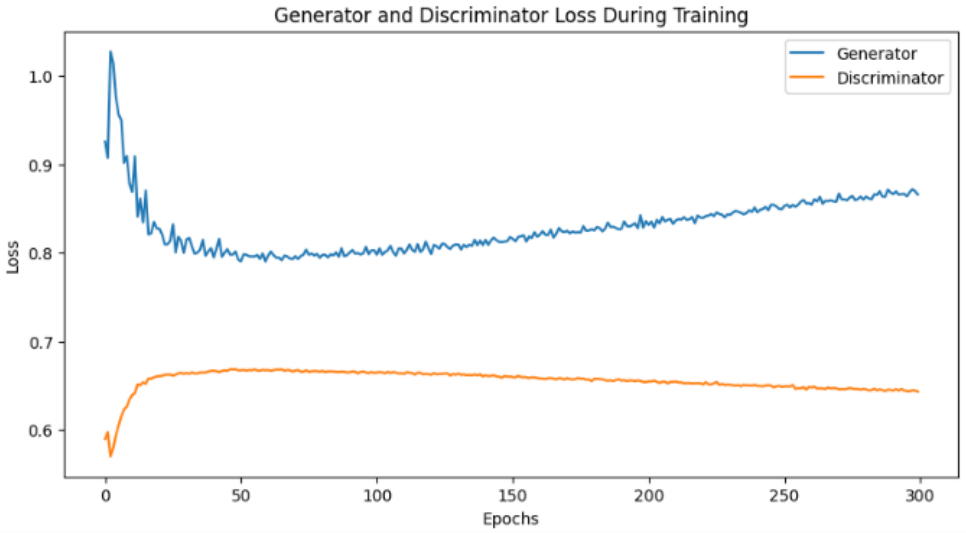


Figure 21: cDCGAN loss over 300 epochs

Algorithm 15 Training Loop for Conditional DCGAN

Setup remains unchanged
for epoch = 1 **to** last epoch **do**
    for each batch $(imgs, labels)$ **in** dataset **do**
        Prepare real and fake labels
        Train Discriminator:
            Train on real images with corresponding labels
            Generate fake images from random latent points and random labels
            Train on fake images with fake labels
            Calculate discriminator loss and update
        Train Generator:
            Generate new fake images with labels
            Calculate loss by passing fake images and labels to discriminator
            Update generator
    end for
    Rest remains unchanged
end for

6 Testing and Evaluation

This section outlines the comprehensive examination of the testing and evaluation conducted on our models, and discusses their performance outcomes. We also conduct a qualitative and quantitative review of the models' image generation.

6.1 Testing Methods

This section outlines a detailed examination of the testing strategies employed during the development of the models. During the development phase, our models underwent rigorous

Figure 22: cDCGAN samples after 300 epochs of training

testing to confirm their functionality and performance. This testing strategy encompassed multiple layers of evaluation:

6.1.1 Sanity checks

We conducted initial sanity checks to ensure basic operational integrity. This included printing the output layer shapes of the U-Net, as well as the Discriminator and Generator in our GAN models, to verify that information was being handled correctly and to preempt issues related to data dimensions and model architecture. In our DDPMs, we created plots for the pre-computed noises to ensure our variance schedulers worked accurately, and tested the forward diffusion process by plotting the images' degradation process.

6.1.2 Visual Inspection

At each epoch of model training, we sampled images to conduct preliminary visual inspections as illustrated in Figure 23. This allowed us to assess the diversity of the generated images and their adherence to expected content standards before proceeding to more formal evaluation methods.

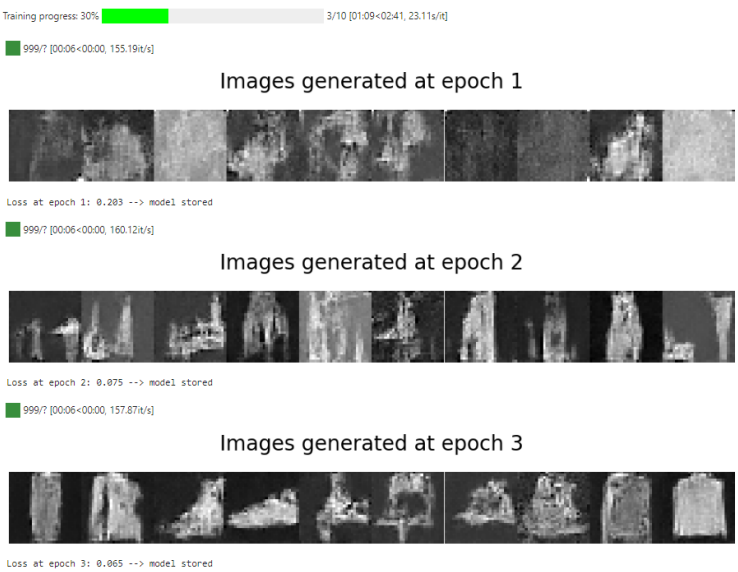


Figure 23: Sampling during training

6.1.3 Loss Monitoring

We plotted the loss of the models against epochs to determine the most suitable epoch for training to achieve convergence. This graphical analysis helped identify optimal stopping points for training and ensured that the models did not overfit or underfit.

Referring to the plotted losses of our models in Figures 15, 17, 19, and 21, we make certain observations:

In the unconditional DDPM, we observe a quick initial decrease in MSE loss, typical to this type of loss function as the model quickly learns to estimate basic features in the data. However, after epoch 100, we notice that the loss reduction rate slows down, indicating that the model is now refining its understanding, and focusing on complex patterns. Despite the decreased rate in the fall of the loss, the graph still exhibits a downward trend. Training continues up to epoch 300 to capture incremental improvements, ensuring the model captures subtle data features and relationships thoroughly. We would like to train for more iterations in future implementations, to match the depth of testing conducted by studies like Dhariwal et al.[3]

The conditional DDPM exhibits a similar loss trend, suggesting that the addition of conditioning does not drastically change the fundamental learning behavior of the model. Both DDPMs benefit from extensive training, which likely enhances their ability to capture the nuanced features of their data distributions. In contrast, the unconditional DCGAN shows well-stabilized losses, especially for the discriminator, indicating a good balance between the discriminator and generator where the generator produces increasingly realistic outputs.

For the conditional DCGAN (cDCGAN) however, the losses initially converge, indicating effective integration of conditioning information and improvement in both the discriminator and generator. However, as training progresses, their losses begin to diverge, potentially signaling overfitting by the generator or more effective discrimination by the discriminator.

Despite this, our conditional DCGAN still outperforms all the other models in terms of FID scores, detailed in the quantitative results of our study (Section 6.3.2). This divergence warrants further investigation and might call for parameter tuning, however this is left to future work due to time constraints.

6.1.4 Performance Benchmarks

During development, we established baseline performance benchmarks. These benchmarks were periodically revisited to compare against the current model performance, ensuring continuous improvement and alignment with the goals of the project.

6.2 Evaluation Strategy

Our evaluation strategy employs both quantitative and qualitative methods to thoroughly assess the performance of our generative models. This approach allows for a comprehensive analysis, combining measurable metrics with subjective assessments to validate the quality of our models' generations.

6.2.1 Quantitative Analysis

Quantitative evaluation begins with calculating the Fréchet Inception Distance (FID) for different model configurations:

- **Initial Baseline:** We first measure FID scores on a sample size of 1,000 images per model to establish a quick baseline. Despite the higher variance due to the small sample size, these initial measurements provide a rapid assessment of model quality.
- **Expanded Evaluation:** To increase reliability, sample sizes are expanded progressively to 10,000 and 50,000 images. To manage computational loads effectively, 50,000 images are sampled in batches of 10,000 across five iterations.

6.2.2 Quantitative Comparison of Scheduling Methods

Special attention is given to comparing the linear versus cosine scheduling methods used in our diffusion models:

- **Scheduling Effectiveness:** We evaluate and compare the impact of linear and cosine noise scheduling on the quality and stability of generated images. This comparison is crucial for justifying the selection of the linear scheduler as the preferred method for our implementations.

6.2.3 Qualitative Analysis

Visual reviews are conducted regularly throughout the training process. Generated images, especially those used for FID calculations, are carefully examined to subjectively assess aesthetic quality and adherence to expected characteristics:

- **Visual Consistency:** Continuous visual inspection helps monitor the consistency and improvement of image quality over training iterations, providing insights that complement quantitative findings.

6.3 Results and Discussion

6.3.1 Linear vs Cosine Schedulers

The performance of the linear and cosine schedulers in the Unconditional DDPM model is compared quantitatively in Table 2 below, focusing on their respective training times and Fréchet Inception Distance (FID) scores over 50,000 samples after 300 epochs of training. The results indicate a marginal performance difference between the two scheduling methods. The unconditional DDPM with a linear scheduler achieves a slightly better FID score of 26.14 compared to the 26.50 scored by the cosine scheduler. Additionally, the linear scheduler also requires slightly more training time, a difference of approximately two minutes over 300 epochs.

Table 2: Quantitative Results of Schedulers

Model	Training Time (300 epochs)	FID Score (50k samples)
Unconditional DDPM (Linear)	1:22:40	26.14
Unconditional DDPM (Cosine)	1:20:51	26.50

Despite the close scores, the superior FID performance of the linear scheduler, albeit small, alongside its straightforward implementation, makes it a preferable choice for our conditional DDPM. The simplicity of the linear approach, combined with its marginally better performance, supports its use in scenarios where a clear, uncomplicated method is advantageous. Thus, we have opted to implement the linear scheduler in our conditional DDPM due to its efficacy and simplicity, even though the performance difference is almost negligible at these high FID scores.

The slight superiority of the linear scheduler over the cosine scheduler in our DDPM could stem from various factors. The linear scheduler's straightforward, predictable progression of noise levels might provide a more stable and consistent training experience, potentially aligning better with the model's learning dynamics and the inherent data distribution. This simplicity could also facilitate easier hyperparameter tuning and more effective navigation of the error surface, leading to better convergence properties. Furthermore, the constant rate of noise decay with the linear approach could inherently suit the training regime employed, compared to the variable noise adjustments of the cosine scheduler. We speculate that the cosine scheduler might provide better results than the linear scheduler when a lower number of diffusion timesteps is implemented. Finally, it is also possible that our implementation of the cosine scheduler may not be fully optimized or correctly implemented (though our tests do not indicate this), a limitation we acknowledge but could not explore further due to time constraints. These aspects, alongside a deeper investigation into the scheduling methods' impact on model performance, is left as a recommendation for future work.

6.3.2 Quantitative Results

The quantitative outcomes are summarized in the table below, providing a clear comparison of model performance across different configurations and scheduling methods.

Referring to the findings by Dhariwal et al.[3], where diffusion models outperformed GANs in various aspects. Our findings diverge somewhat, possible due to several factors unique to our implementation and experimental setup.

Table 3: Quantitative Results of Model Performance

Model	Training Time (300 epochs)	FID Score (50k samples)
Unconditional DDPM (Linear)	1:22:40	26.14
Conditional DDPM	1:22:43	21.11
Unconditional DCGAN	1:45:15	14.76
Conditional DCGAN	1:57:30	11.46

Firstly, the U-Net architecture employed in our diffusion model might not possess the depth and complexity to fully capture and generate the nuances of the training data. The absence of advanced components/improvements like self-attention blocks, which have previously proven to be effective in enhancing the model's perceptual capabilities, could limit the quality of the generated images. Our simpler U-Net may contribute to the residual noise observed in some of the DDPM's generated images, adversely affecting the FID scores, which is discussed further in the Qualitative Results (Section 6.3.3). In Figure 7, we see how different types of noise, such as Gaussian and salt and pepper noise, significantly impact FID scores. This sensitivity to noise suggests that even minor residual noise in our diffusion model's output could lead to a significant increase FID scores.
Moreover, the training time versus image quality trade-off is another crucial aspect to consider. Our experiments reflect this trade-off, where the GANs, despite longer training durations, outperformed the faster-training diffusion models in terms of FID scores.

Another point of consideration is our use of a guidance scale of 3.0 in our conditional DDPM. According to the Classifier-Free Guidance paper [4], there exists a trade-off where FID scores are negatively affected as the guidance scale increases possibly due to lower diversity in generated images. This suggests that a lower scale might yield better FID results, a hypothesis left to future work for verification.

Considering these aspects, our future work should explore a more complex U-Net architecture, potentially incorporating deeper layers and self-attention mechanisms, to enhance the model's ability to generate noise-free images. Additionally, fine-tuning the guidance scale

in the Classifier-Free Guidance model and further optimization of our model's parameters could provide clearer insights into the capability of the diffusion models. This comprehensive future investigation will aim to reconcile our findings with established research and explore ways to leverage the inherent strengths of diffusion models more effectively.

We also note that the traditional DDPM models we developed had to step through the entire reverse diffusion process every time we wished to sample images. This led to the sampling times of the diffusion models being significantly longer than the sampling times of DCGANs. Motivated by this discrepancy in sampling times, we wish to investigate the relevant literature on diffusion models that incorporate techniques aimed at decreasing sampling times.

6.3.3 Qualitative Results

This section provides a qualitative assessment of the visual quality and diversity observed in the images generated by our diffusion and GAN models. The review focuses on specific characteristics that could be influencing the performance metrics and overall image fidelity.

Diffusion Models

- Residual Noise: Images generated by diffusion models often exhibit residual noise, particularly noticeable in the backgrounds of clothing items. This points to a gap in how the diffusion model understands the data distribution, where the model fails to recognize that backgrounds should be uniformly black and noise-free. Such residual noise is more prevalent in unconditional DDPMs and less so in class-conditioned diffusion, indicating that class conditioning helps mitigate this issue to some extent. Refer to Figure 26.

- Coherence: Unconditional diffusion models occasionally struggle to maintain coherent shapes for clothing items. On the other hand, conditional diffusion models consistently produce well-defined and coherent shapes. This improvement in shape coherence in the conditional models can be attributed to the effective use of class label embeddings

Figure 24: **Comparison of Generated Diffusion Images (300 Epochs of Training)** – Unconditional DDPM (**left**) and Classifier-Free Guidance DDPM (**right**).

and the application of a higher guidance scale in Classifier-Free Guidance [4].

- Diversity: There appears to be less diversity in the images sampled from the conditional diffusion model as seen in Figure 6.3.3. This reduced variability could be impacting the FID score, suggesting a potential trade-off between diversity and noise control through the guidance scale.

Generative Adversarial Networks (GANs)

- Background/Noise Control: Unlike our DDPMs, GANs do not exhibit issues with residual noise and consistently achieve a clean, black background in almost every generated sample. This clean background significantly contributes to their lower FID scores and can be seen in Figure 25.

- Coherence: Unconditional GANs occasionally produce images that lack a coherent shape or clear concept of the clothing item. In contrast, conditional GANs consistently generate well-formed clothing items, underscoring the effectiveness of label embeddings in improving shape coherence through learned class-specific features.

- Diversity and Detail: Conditional GANs display a notable diversity in generated images, sometimes producing distinguishable textures, graphics, and depth in the generated images of clothing (as well as they can in 28x28 pixels). This is a likely factor in their superior FID scores.

- Artefacts: Despite their strengths, GANs sometimes generate images with noticeable artefacts, such as missing parts of clothing items, such as a T-shirt missing a sleeve or sandals missing straps. This is shown in Figure 27. Note that this issue occurs in both the unconditional and conditional GAN, however it is more prevalent in the unconditional version.

Comparative Insights Comparing these models, it's evident that each has its strengths and weaknesses. While GANs excel in achieving clear backgrounds and coherent shapes with diverse details, they suffer from occasional artefacts. Diffusion models, while struggling with residual noise and less diversity, benefit from class conditioning to improve shape coherence and do not exhibit artefacts similar to the GAN generations. These observations suggest

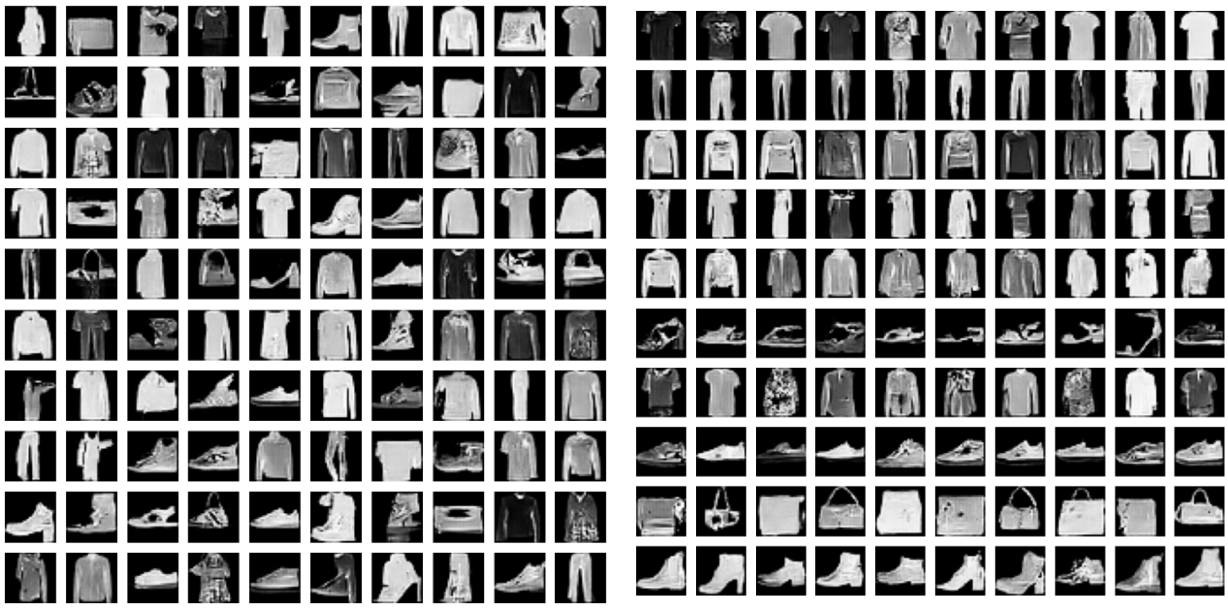


Figure 25: **Comparison of Generated DCGAN Images (300 Epochs of Training** – Unconditional DCGAN (**left**) and Conditional DCGAN (**right**).

Figure 26: Unconditional and Conditional DDPM generations with residual noise

Figure 27: DCGAN and CDCGAN generations with missing parts

areas for future refinement, particularly in enhancing noise learning within the diffusion models and reducing artefacts in images generated by the DCGANs.

This qualitative review discusses the trade-offs involved in model selection and optimization, offering a subjective assessment of each model's performance beyond quantitative metrics. Future work will focus on addressing these identified issues to enhance both the aesthetic qualities and the technical performance of the models.

6.4 Concluding Evaluation

In this comprehensive evaluation, both quantitative and qualitative analyses have revealed nuanced insights into the capabilities and limitations of our generative models. The linear scheduler in diffusion models marginally outperformed the cosine variation, affirming its suitability for further use due to its simplicity and stability in training. Qualitatively, diffu-

sion models exhibited residual noise issues, having a definite negative impact on their FID scores, while GANs displayed superior noise control and lower noise levels, leading to better quantitative outcomes despite occasional image artifacts. These findings highlight the importance of model architecture and parameter settings, suggesting potential enhancements such as integrating advanced features like self-attention blocks in diffusion models to reduce noise and improve image clarity. Our future optimizations will focus on refining these models, attempting to improve their application in practical settings and improve performance across diverse scenarios. The insights gained emphasize the potential of both model types to produce high-fidelity images, guiding our future developments to leverage their strengths more effectively.

7 Conclusions and Future Work

The completion of this study provides several pathways for further research and enhancements. These recommendations are informed by the limitations encountered and the potential we have observed during our investigation.

7.1 Conclusion

This dissertation has presented a detailed comparative analysis of two dominant paradigms in the field of generative modeling: Diffusion models and Generative Adversarial Networks (GANs). By developing foundational models and extending these with class labels, we have not only explored their capabilities in generating high-fidelity images but also assessed their performance under constrained computational resources. The study's findings provide insights into the architectural complexities, performance metrics, and practical limitations of these cutting-edge image generation technologies.

Our quantitative and qualitative analyses demonstrate that while both model types have their strengths and weaknesses, certain distinctions are clear. Diffusion models excel in generating detailed and diverse images without requiring adversarial training setups, which simplifies the training process and potentially reduces the propensity for mode collapse. On the other hand, GANs have showcased their robustness in generating visually coherent and less noisy images, without the need for Markovian chains [1], though they are more susceptible to training instability.

The project's constraints, limited by single-GPU capabilities and the simplicity of the FashionMNIST dataset [6], have not hindered the depth of our inquiry but rather highlighted the feasibility of conducting AI research within common resource limits. Our study contributes to the broader discourse on scalable and efficient AI, pushing the boundaries of what can be achieved in generative modeling with minimal resources.

Furthermore, the adaptation of project methodologies from a traditional Waterfall to an Agile approach mid-project underscores the dynamic nature of research and development in computer science. This adaptive strategy not only accommodated unforeseen challenges but also enhanced the iterative process of model refinement, aligning the project management approach with the technical endeavors. For more information on how the project was managed, refer to Section 8.

As we conclude our research, it is evident that the journey of enhancing and optimizing generative models is far from complete. The insights gained within this study lay the groundwork for further exploration and refinement, with numerous pathways for future research to build upon the foundations established in this study.

The next section on Future Work, outlines specific recommendations and potential projects that can further advance the capabilities of generative models, aiming to address the limitations encountered within this study.

7.2 Recommendations for Further Research

We would like to explore increasing the complexity of the models and applying them to a broader range of datasets, including those with higher resolution and RGB images. This would test the models' robustness and provide more insights into their performance in varied contexts.

Furthermore, we also intend to investigate the Denoising Diffusion Implicit Models (DDIM) paper [15] to incorporate faster sampling techniques within our DDPM framework. Faster sampling could address some of the time constraints we encountered in the project, potentially making the diffusion process more practical for real-world applications where computational resources and time are limiting factors. Building on the faster sampling DDPMs, we would also like to explore the concept of learned variances as proposed by Nichol et al. They noted "we find that learning variances of the reverse diffusion process allows sampling with an order of magnitude fewer forward passes with a negligible difference in sample quality"[11].

We would like to refactor the codebase for greater modularity, allowing for seamless adaptation to different datasets and experimental conditions in the future.

7.3 Proposed Future Projects

We would like to build on our gained knowledge of diffusion models, by pursuing the development of text-guided generative models using the CLIP (Contrastive Language–Image Pre-training) framework [16]. CLIP has demonstrated significant potential in filling the gap between textual descriptions and producing images [16]. By integrating CLIP within our diffusion model, we could create models that are capable of generating highly accurate and context-based images based on textual input from a user. This initiative would enhance the capabilities of our generative systems.

Enhancing the resolution of diffusion model outputs through upsampling techniques is a promising path to improving the quality of our generated samples. We would like to investigate the potential of cascaded diffusion models [17], which concatenate multiple diffusion models together, where the first is tasked with producing an image and the subsequent models are tasked with upsampling that image to a higher resolution.

Venturing beyond image synthesis, the exploration of diffusion models and GANs in the generation of multimodal media such as audio and video content is intriguing. This expansion is ambitious due to the complex nature of temporal data in audio and video, however we believe it holds considerable promise for transformative applications in the future.

7.4 Speculation on the integration of GANs and Diffusion Models

A point brought up by the project supervisor, was to consider the possibility of the incorporation of elements from diffusion models and GANs into a singular architecture.

In theory, combining the strengths of both GANs and diffusion models could potentially

lead to a model that leverages the fast generation capabilities of GANs and the stability of diffusion models. Such an integration could take various forms, such as utilizing a diffusion model to refine or upsample outputs initially generated by a GAN, or perhaps employing adversarial training techniques to enhance the training process of diffusion models.
Looking at recent literature regarding the integration of these models, we found that there had been significant strides in integrating these two approaches. Dhariwal et al. in their seminal work "Diffusion Beats GANs"[3] demonstrate that diffusion models, when guided by classifiers, can achieve state-of-the-art results in conditional image synthesis tasks. They report that "with classifier guidance, we find that we can sample with as few as 25 forward passes while maintaining FIDs comparable to BigGAN" [3]. This approach significantly reduces the number of required sampling steps, addressing one of the primary limitations of traditional DDPMs which is their slow sampling rate. Furthermore, we note that the use of a classifier to guide image generation, somewhat resembles the adversarial training process of GANs.

Another groundbreaking approach is presented in the work titled "Diffusion-GAN" [18], which introduces a novel GAN architecture that uses a Gaussian mixture distribution defined over diffusion steps. This method injects instance noise for stable GAN training, where "A random sample from the mixture, which is diffused from an observed or generated data, is fed as the input to the discriminator" [18]. The authors provide a theoretical analysis that underscores the soundness of Diffusion-GAN, highlighting its capability to deliver stable and data-efficient training across diverse datasets. They detail a model that consistently improves the synthesis of photorealistic images.[18]

Intrigued by the findings of these papers, we see the possibility of several innovative pathways for future research. Utilizing diffusion processes to refine outputs generated by GANs can potentially mitigate common issues such as mode collapse, thereby enhancing the diversity and realism of the generated images. Moreover, the integration of adversarial loss functions within the diffusion model's training framework could accelerate convergence and improve the fidelity of generated samples, borrowing the quick feedback mechanisms typical to GANs.

Employing the discriminator from GANs to guide the diffusion steps could also work to optimize the generative process, merging the detailed feature extraction capabilities of GAN discriminators with the generative prowess of DDPMs. Motivated by the learnings from this project, we are committed to deepening our understanding and advancing the field through continued research and study in the future.

8 Project Management

8.1 Development Methodology

8.1.1 Initial Approach

The project was initiated with a well-defined scope and a sequential timeline, which aligned well with the Waterfall methodology's structured nature. This traditional approach was particularly suited to the project's initial requirements, which were stable and clearly outlined, with strict deadlines. The methodology facilitated a systematic progression through the project's phases, with strict weekly deliverables set from the outset.

8.1.2 Challenges and Unforeseen Obstacles

One significant constraint during the development of the models, was the inability to continue model development and training during periods away from the primary workstation, such as during university breaks. As a result, we shifted our focus to further research into generative models and the surrounding field during this time.

Despite these limitations, the development environment supported the project's agility and adaptability. By prioritizing research during downtime and actively developing when resources were available, the project maintained momentum and continued to progress towards its objectives.

However, an unexpected need for a temporary withdrawal from university due to medical reasons significantly disrupted the planned progress and timeline. This interruption necessitated a critical reassessment of the project's scope and deadlines upon my return to university, leading to substantial adjustments in the project management approach discussed in the following section.

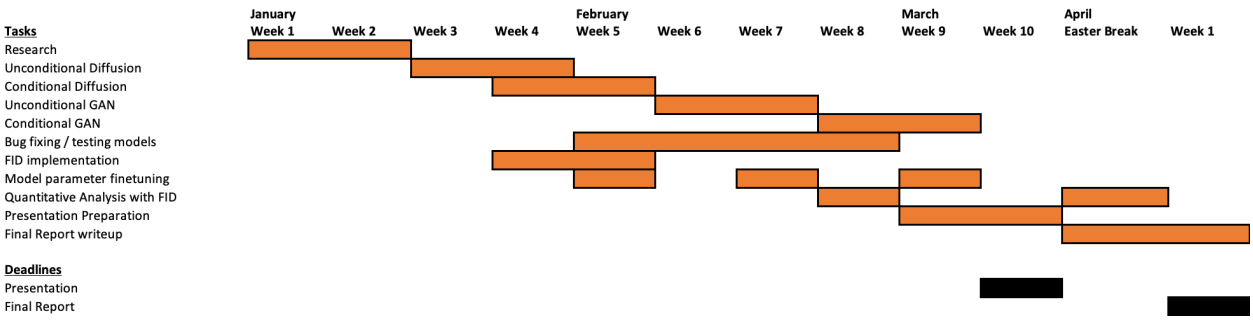


Figure 28: Gantt Chart outlining Term 2 and Term 3 deliverables

8.1.3 Adaptations in the development methodology

Upon resuming the project it became apparent that a more flexible and adaptive approach was required to manage the altered project dynamics effectively. The Agile methodology was thus adopted, which is renowned for its flexibility and responsiveness to change. This method allowed for iterative and incremental development, where the project could evolve in response to immediate feedback and ongoing revisions to the scope. Weekly deliverables were redefined (in recurring meetings with the project supervisor) based on current priorities and progress, allowing for continuous adaptation, and ensuring that the project remained on track despite the reduced timeline.

The transition to Agile proved invaluable, enhancing the project's ability to swiftly adapt to unforeseen changes and new requirements. It ensured that the project could continue moving forward despite significant interruptions, ultimately supporting the achievement of a minimum viable product (MVP) within the constrained timeframe.

We employed a Gantt chart, seen in Figure 28, that was continuously updated to reflect new deliverables or modifications to existing functionalities. The support of the project supervisor was instrumental in reviewing the state of the project and clearly defining these deliverables. We also employed a Kanban board for weekly task management, which was a key Agile practice that allowed for flexibility and responsiveness to the changing needs of the project.

8.2 Version Control

We managed Version control and incorporated iterative development through GitHub, which offered a systematic approach to tracking changes and managing revisions in the code.

9 Author's Reflections

This work has expanded my horizons, transitioning a budding interest into a rigorous academic project despite challenges like computational limitations and health issues. Overcoming these obstacles required a significant reduction in scope and adaptations in my approach and methodologies, which became a profound learning experience. This process deepened my understanding of a new field, demanded extreme dedication, and shaped my Computer Science degree by illustrating how practical applications of theory highlight the value of innovation under constraints. This endeavor stands as a testament to how focused and driven research can yield insights and lay the foundation for future research.

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[2] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," 2015.

[3] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," 2021.

[4] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022.

[5] 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.

[6] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[7] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016.

[8] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[10] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," 2017.

[11] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[12] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[13] P. Brian, "Ddpm." `https://github.com/BrianPulfer/PapersReimplementations/blob/main/src/cv/ddpm/models.py`, 2023.

[14] J. Brownlee, "How to develop a conditional gan (cgan) from scratch." https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/, Sep 2020.

[15] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," 2022.

[16] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021.

[17] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans, "Cascaded diffusion models for high fidelity image generation," 2021.

[18] Z. Wang, H. Zheng, P. He, W. Chen, and M. Zhou, "Diffusion-gan: Training gans with diffusion," 2023.