Analysis of Noise Addition Times in Diffusion Models and the Application of Generated Images

Hongwei Zhu, MSc.
School of Computing and Communications, Lancaster University
A dissertation submitted for the degree of *Master of Science* in Data Science.
September, 2023

Abstract

Since the debut of ChatGPT and Stable Diffusion in the second half of 2022, AIGC's exceptional creative capabilities have not only attracted researchers in the scientific community but also garnered widespread societal attention. In this paper, we delve deeply into the application of deep learning in the field of image generation. We first provide an overview of the fundamental principles of deep generative models, followed by a detailed examination of several classic deep generative models represented by GANs, outlining their core principles, optimization strategies, and respective strengths and weaknesses. Subsequently, we focus on the latest technology in the image generation domain — the diffusion model. We provide an in-depth analysis of the working principles of the DDPM diffusion model, including its forward process, denoising procedure, training strategy, and neural network architecture. Although the diffusion model surpasses GANs in terms of image quality and diversity, its generation speed is relatively slow due to the need for multiple passes through the neural network for denoising during the generation process. In this study, we investigate the impact of the number of noise additions on the image quality produced by the diffusion model and attempt to train image classification neural networks using images generated by the diffusion model. Based on our research findings, we conclude that: (1) In the DDPM model, when employing a linear strategy for adding Gaussian noise, the optimal image quality is achieved when the noise is added 800-900 times; further increasing the number of noise additions does not lead to any further improvement in image quality. (2) Using images generated by the diffusion model for data augmentation in image classification datasets effectively enhances the accuracy of image classification neural networks.

Acknowledgements

First and foremost, I would like to extend my deepest gratitude to my advisor, Professor Richard Jiang, for his invaluable guidance and unwavering support throughout my academic journey. It is his patience, passion, and expertise that have illuminated my path. I feel immensely fortunate and proud to have such a distinguished and dedicated mentor.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 2012, AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) achieved a milestone victory in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Deng et al., 2009), heralding the dawn of a new era in deep learning. This accomplishment promptly positioned deep learning at the forefront of artificial intelligence research. With the ever-expanding datasets and the robust computational capabilities of GPUs, scholars and researchers successively crafted more intricate and deeper neural network architectures. These innovative models showcased remarkable performance across various domains, such as computer vision, natural language processing, recommendation systems, and autonomous driving, often significantly outpacing traditional machine learning approaches like random forests, SVMs, and regression analysis.

Deep learning, an advanced form of machine learning, aims to analyze and interpret data through intricate neural network architectures. These networks typically consist of multiple layers, each containing a vast number of neurons. Data starts from the input layer, is sequentially processed through various hidden layers, and eventually reaches the output layer. To enhance the expressive power of the neural network, activation functions such as ReLU and sigmoid are introduced in the hidden layers, enabling non-linear data transformations and model expansions. Once an appropriate dataset and network architecture are established, the network's loss function is continually optimized using backpropagation and gradient descent. Beyond traditional fully connected networks, there are structures designed specifically for particular tasks. For instance, Convolutional Neural Networks (CNNs) are tailored for tasks like image classification and segmentation, while Recurrent Neural Networks (RNNs) excel in NLP tasks, including text translation and speech recognition. Recently, the Transformer architecture equipped with attention mechanisms has surpassed the performance of RNNs in the domain of NLP, with the achievements of ChatGPT (Radford, Narasimhan, et al., 2018; Radford, J. Wu, et al., 2019; Brown et al., 2020; OpenAI, 2023a) serving as a prime example.

In contemporary deep learning research, deep generative models undoubtedly stand out as a prominent and vibrant subfield. These models endeavor to grasp and learn the underlying

distribution of data, enabling the generation of new samples that bear similarities to the training data yet remain novel. Compared to discriminative models that merely classify or recognize data, deep generative models delve into capturing the essence and richness of data. Notably, GAN (Goodfellow et al., 2014) and VAE (Kingma and Welling, 2013) are the luminaries in this realm, demonstrating remarkable potential and achievements in image synthesis, audio emulation, and text generation. The success of these models hinges on their ability to accurately capture and replicate intricate data distribution characteristics in high dimensions. Beyond their academic and technical contributions, deep generative models have showcased their prowess in practical scenarios, bringing revolutionary innovations and new possibilities to domains like artistic creation, data simulation, and drug design.

In recent times, the diffusion model (Ho, Jain, and Abbeel, 2020; Luo, 2022), grounded in the principles of non-equilibrium thermodynamics (Sohl-Dickstein et al., 2015), has ascended to a dominant position in the realm of deep generative models, thanks to its exemplary generative capabilities. When evaluating both the quality of generated images and the richness and variety of their content, the diffusion model distinctly outshines GANs. At its heart, the diffusion model adopts a clear and concise methodology: it progressively infuses Gaussian noise into an image, aligning it with a Gaussian distribution. Throughout this procedure, the model meticulously captures the characteristics of the noise introduced at each iteration. After the training concludes, this model can adeptly produce images reminiscent of the training set from a specified Gaussian noise.

A significant difference between the diffusion model and GAN is that the generation process of the diffusion model requires multiple steps. Random Gaussian noise can be directly transformed into images through the neural network of a GAN. In contrast, the diffusion model requires a multi-step denoising process. Each time an image is input into the diffusion model, it predicts the noise to be added to the image. After removing the predicted noise from the image, it is re-input into the network. This process is repeated multiple times until the desired image is generated. In this paper, we experimented with different numbers of this process, comparing the image generation time with the quality of the generated image.

Furthermore, we used images generated by the diffusion model as a data augmentation method for the classification neural network. We compared three scenarios: using only the training set images, using only the synthesized dataset images, and using an enhanced dataset that combines both the original and synthesized images.

From the experiments in the paper, we derived two core conclusions:

1. When training the DDPM diffusion model using a dataset, the number of noise additions significantly affects the quality of the generated images. As the number of noise additions increases, the image quality gradually improves. However, after reaching a certain level, the improvement plateaus. Different datasets achieve different optimal quality levels.

2. After the diffusion model is trained, using its generated images as training data for the classification neural network yields good results, although not as good as using the

original training set directly. However, when we supplement the original dataset with images generated by the diffusion model and apply data augmentation, the performance of the classification neural network can be further enhanced.

In this paper, in Chapter 2, we delve deeply into the concept of deep generative models, their primary differences from discriminative models, and the core problems they aim to address. We first provide an overview of the most common deep generative models, such as GAN, and then detail VAE and explore how it evolves step by step into the diffusion model. Chapter 3 elaborates on the working principles of the diffusion model, including its noise addition, denoising, training strategies, and network structure, and provides an in-depth description of our experimental design and methods. In Chapter 4, we present the experimental results through charts and highlight the key findings from these results. Chapter 5 offers a thorough analysis and discussion of the experimental results, including the reasons behind the results, the theoretical foundation, and the potential impacts they might bring. Chapter 6 concludes the paper, where we discuss the significance of the experimental results from a macro perspective, their impact on the field, and the directions for future research.

Chapter 2

Background

2.1 Discriminative Model and Generative Model

Deep discriminative models primarily aim to classify or differentiate between data categories, while deep generative models strive to understand and generate data that mimics a given dataset's structure. Famous generative models include Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs).

Discriminative models are tailored for tasks that require sharp categorization. They function by understanding the differences between categories and fine-tuning their decision boundaries. These models often leverage rich feature hierarchies and have architectures that can efficiently capture and represent complex patterns within data, making them especially valuable in tasks such as image or speech recognition where discerning fine-grained details is crucial. However, their performance hinges significantly on having a wealth of labeled data. In situations where labeled data is sparse or when encountering unfamiliar data points, they might not perform optimally.

Generative models, conversely, focus on capturing the intrinsic structure and distribution of the data. They are not merely content with understanding what separates categories but go a step further to understand what constitutes each category. Their ability to generate novel, coherent data samples makes them valuable in tasks where data augmentation or synthesis is needed. These models operate by learning the joint probability distribution of the data, and they can provide insights into the data's latent structures and correlations. Yet, training them tends to be a more intricate affair, often necessitating more sophisticated techniques to ensure that the generated samples remain authentic and meaningful.

Both discriminative and generative models offer distinct advantages in their respective domains of application. Discriminative models excel in sharp categorization tasks, while generative models shine in creative endeavors and scenarios where understanding the data's essence is pivotal. Together, they represent two fundamental approaches in deep learning, each offering unique perspectives on data understanding and manipulation.

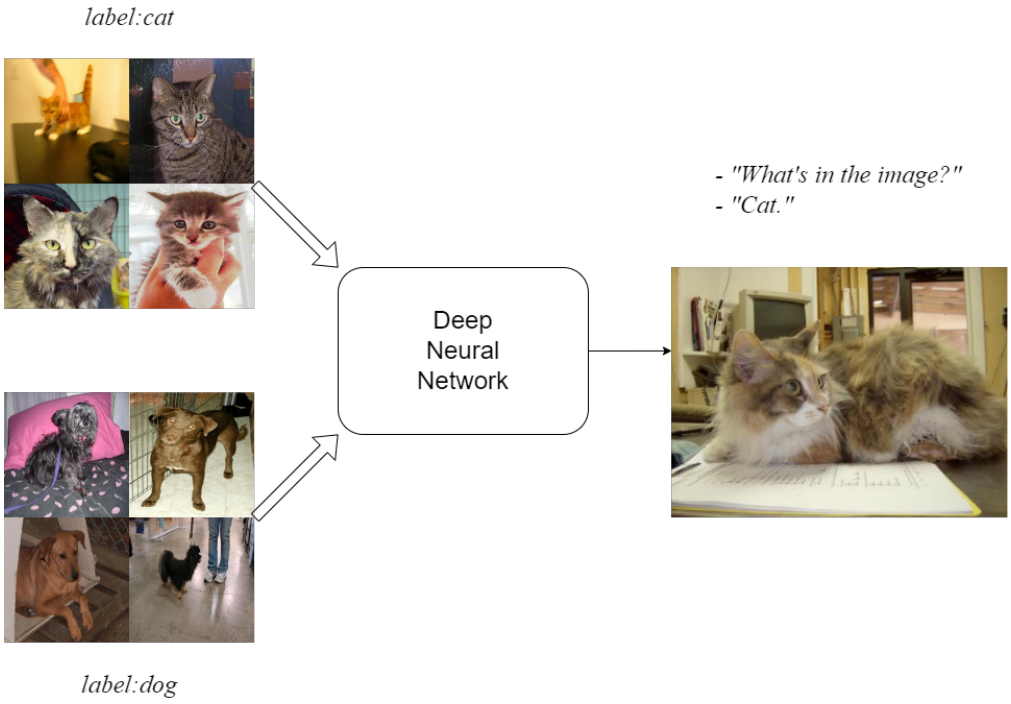

Figure 2.1: An image classifier is a discriminative model. During its training phase, the dataset employed comprises images and their corresponding labels. In the inference stage, this neural network takes an image as input and predicts the classification label for the image.

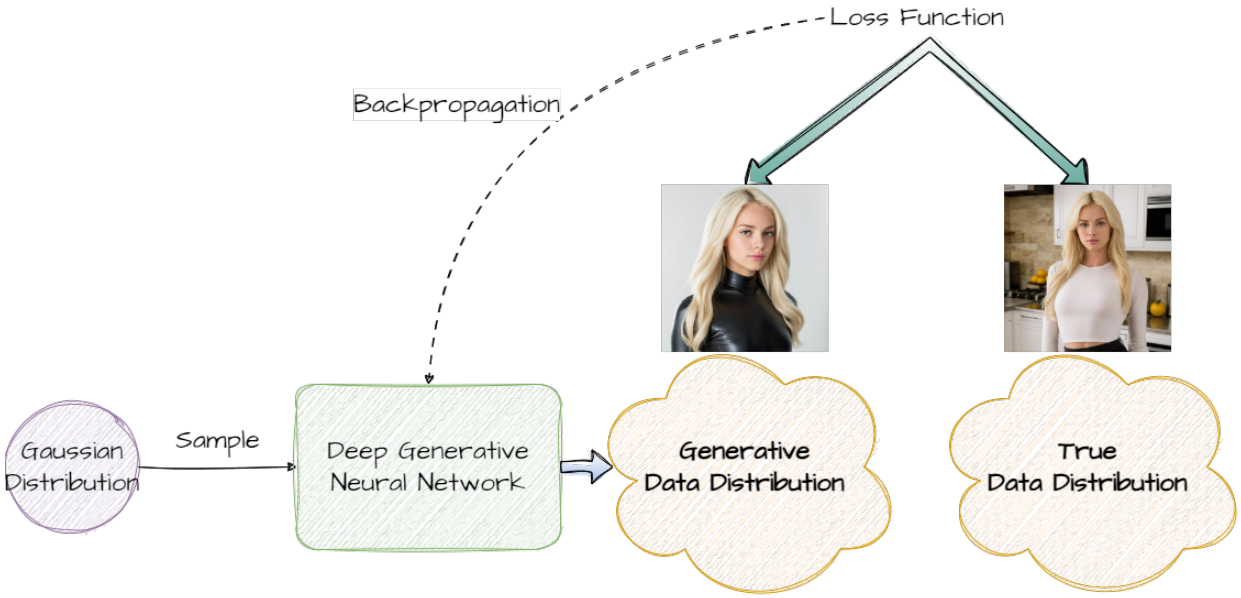

Figure 2.2: Generative models transform a Gaussian distribution into another probability distribution, ensuring its closeness to the training set distribution through a loss function (OpenAI, 2023b).

2.2 Autoregress Model

The autoregressive model (Cho et al., 2014; Oord, Kalchbrenner, and Kavukcuoglu, 2016; Salimans et al., 2017), in the domain of deep generative modeling posits a sequence by modeling the probability of each data point based on its predecessors. Mathematically, for a sequence $\mathbf{x} = [x_1, x_2, \ldots, x_T]$, the joint probability distribution can be factorized as:

$$P(\mathbf{x}) = \prod_{t=1}^{T} P(x_t | x_{<t}) \tag{2.1}$$

where $x_{<t}$ represents all preceding elements in the sequence. The conditional probabilities $P(x_t|x_{<t})$ are modeled using intricate deep neural network architectures to capture intricate patterns and dependencies inherent in the data.

Such a factorization approach, while elegant in principle, poses computational challenges in practice. The sequential nature of data modeling, be it per pixel for images or per word for texts, incurs substantial computational overhead during the generation process. Since the computation is inherently sequential, it lacks the ease of parallelism found in other deep generative models, becoming a bottleneck when rapid sampling is essential. Furthermore, relying on preceding data points to generate subsequent ones means that any errors or noise introduced early in the generation process can propagate, adversely impacting the quality

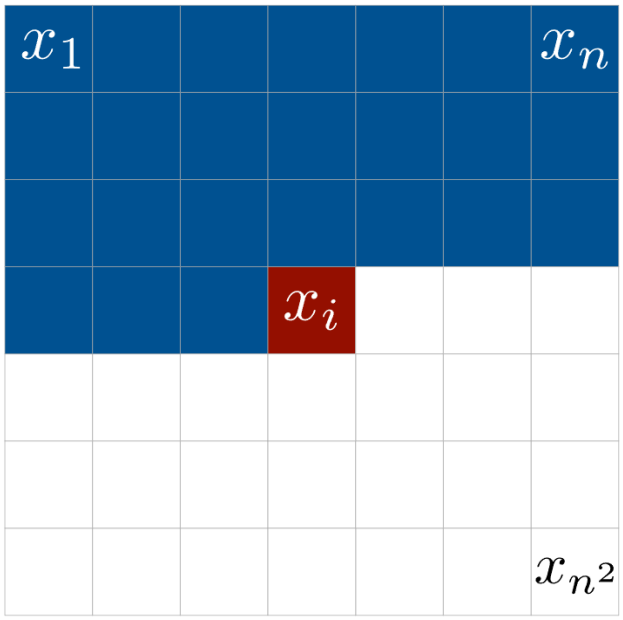

Figure 2.3: The process of generating images with an autoregressive model.

of the generated sequences. As shown in Figure 2.3, the autoregressive model sequentially generates an image pixel by pixel. Beginning from the left of the top row, it progresses rightward and, upon completing a row, shifts to the left of the next one. This procedure continues until the image is fully generated. The regions in blue represent the pixels that have been generated, the white areas denote pending sections, and x_i signifies the currently processed pixel.

Another limitation is their inherent difficulty in considering bidirectional contexts, unlike other models that can capture dependencies from both past and future tokens simultaneously. This becomes particularly pronounced when dealing with languages or data types where context is not strictly unidirectional. Therefore, while the autoregressive property is intuitively appealing, a series of trade-offs must be considered in its adoption.

2.3 Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a class of machine learning models. They consist of two neural networks, the generator and the discriminator, trained simultaneously through adversarial processes. The generator aims to produce data resembling real samples, while the discriminator's goal is to distinguish between actual and generated data. During training, the generator improves its ability to deceive the discriminator, and in turn, the discriminator enhances its capability to distinguish real from fake. This

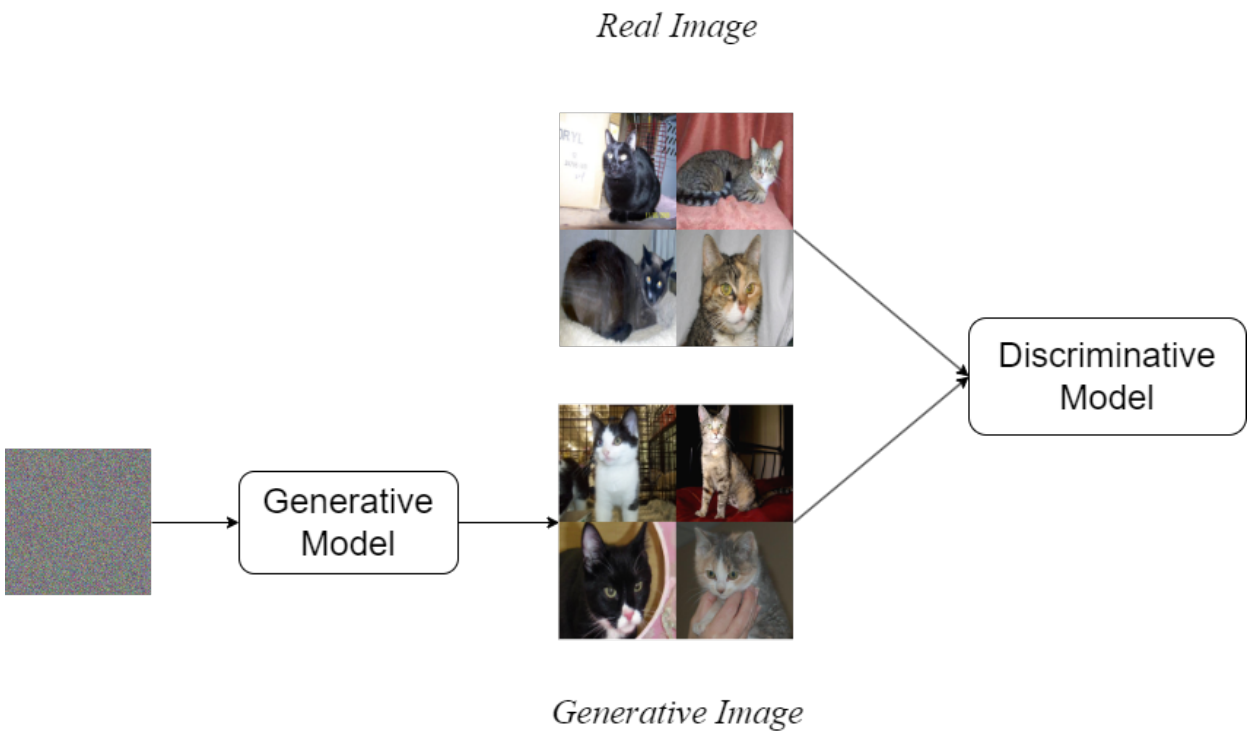

Figure 2.4: The structure of GAN consists of a generative model and a discriminative model. The generative model creates images from noise, while the discriminative model is responsible for distinguishing whether images are real or synthesized by the generative model. During the training process, the two models collaborate closely. Ultimately, the generative model is capable of producing images that are nearly indistinguishable from real ones.

adversarial process leads to the generator creating highly realistic data. The objective of GANs can be represented using the following min-max formulation:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{2.2}$$

Where $\mathbf{x}$ are real data samples drawn from the true data distribution p_{data}, $D(\mathbf{x})$ is the discriminator's estimate of the probability that the real data instance $\mathbf{x}$ is genuine. $\mathbf{z}$ are random noise samples from distribution $p_{\mathbf{z}}$, typically a standard normal distribution. $G(\mathbf{z})$ is the data generated by the generator using noise $\mathbf{z}$, and $D(G(\mathbf{z}))$ is the discriminator's estimate that a generated instance is genuine. GANs have revolutionized tasks in image generation, style transfer, and more. However, they can also pose challenges, including mode collapse and training instability, leading to various refined GAN architectures over the years.

2.4 Variational Autoencoder (VAE)

2.4.1 Variational Inference

Variational inference is a powerful technique used for approximating complex integrals and intractable probabilistic posterior distributions. This method's genesis stems from the field of statistical physics, but it has been widely adopted in modern machine learning, primarily in Bayesian deep learning and graphical models. The principal idea behind variational inference is to frame the problem of posterior estimation as an optimization problem, where the goal is to find the distribution that best approximates the true posterior distribution, subject to certain constraints.

In traditional Bayesian methods, one often wants to calculate the posterior $p(z|x)$ of latent variables z given observed data x. However, for complex models, this direct computation can become infeasible due to the high-dimensional integrals involved. Variational inference sidesteps this challenge by introducing a simpler, parameterized family of distributions, often denoted as $q(z;\lambda)$, where λ are the variational parameters. The main goal is then to adjust λ so that q is as close as possible to the true posterior $p(z|x)$.

Closeness between the true posterior and the approximating distribution is measured using the Kullback-Leibler (KL) divergence. The optimization problem hence becomes the minimization of the KL divergence between q and $p(z|x)$. However, directly minimizing this is still challenging because it involves the intractable true posterior. Therefore, variational inference aims to maximize a lower bound to the log-likelihood of the observed data, known as the Evidence Lower BOund (ELBO). The ELBO is the difference between the expected log likelihood under the approximation q and the KL divergence between q and the prior $p(z)$.

One of the strengths of variational inference is its scalability. Through stochastic variational inference, one can scale the method to large datasets by processing small mini-batches of data and updating the variational parameters in a stochastic gradient ascent fashion. Moreover, by leveraging reparameterization tricks, like the one employed in the Variational Autoencoder (VAE) framework, we can ensure more stable optimization and richer approximating distributions.

However, variational inference is not without its limitations. The choice of the variational family q can be crucial. Too simple a family might not capture the complexities of the true posterior, leading to biased results. Conversely, a too-flexible family might increase computational demands and might not always ensure improved approximations. Additionally, the optimization problem is non-convex, which means multiple local optima can exist, potentially leading to different results depending on initialization and optimization procedures.

In summary, variational inference is an essential tool in the modern machine learning toolkit, offering a scalable and flexible approach to Bayesian inference in complex models. While it has enabled significant advancements, the method requires careful consideration regarding the choice of approximating distributions and optimization techniques to achieve

the best results.

2.4.2 Reparameterization Trick

The Reparameterization Trick is a fundamental technique in deep learning, especially beneficial for variational inference in models like Variational Autoencoders (VAEs). It addresses the intricate challenge of computing gradients concerning the parameters of a stochastic variable. When training models such as VAEs, the goal is to maximize the evidence lower bound (ELBO) relative to the model's parameters. This process involves determining gradients of an expectation over a distribution that is inherently dependent on these parameters. Directly computing this gradient using a naive approach can often be infeasible or result in high variance, leading to unstable or inefficient training.

At its core, the trick aims to decouple parameters from the inherent randomness of the process. Rather than sampling from a distribution that directly depends on the parameters, the approach involves sampling from a fixed, standard distribution. This sample is then transformed using the parameters of interest. For example, in the context of a Gaussian distribution $q(z; \mu, \sigma^2)$, instead of directly sampling z from q, one might sample ϵ from a standard Gaussian distribution $N(0, 1)$ and subsequently compute $z = \mu + \sigma\epsilon$. Here, μ and σ represent the original distribution's parameters, which have been reparameterized in terms of ϵ, a noise term.

The reparameterization trick offers a plethora of advantages. It allows for an analytical gradient concerning the parameters, facilitating backpropagation through neural networks, thereby enhancing training stability and efficiency. Moreover, gradient estimates derived using this trick typically exhibit lower variance compared to other methods, such as score function estimators. The trick's adaptability to mini-batch based stochastic gradient descent, a prevalent method for training deep neural networks on large datasets, further underscores its significance.

However, the technique is not without its limitations. While it's adept for continuous latent variables, it doesn't directly cater to discrete ones, necessitating other techniques like the Gumbel-Softmax trick. Beyond VAEs, the reparameterization trick finds utility in Bayesian Neural Networks, which introduce weight uncertainty, Normalizing Flows that construct intricate distributions by transforming simpler ones, and Stochastic Variational Inference, optimizing variational parameters in a broader spectrum of models. As deep learning continues its evolutionary trajectory, the reparameterization trick emphasizes the synergy between probabilistic modeling and deep learning, heralding innovative model architectures and capabilities.

2.4.3 Principles and Architecture

A Variational Autoencoder (VAE) (Kingma and Welling, 2013) stands as a seminal advancement in the intersection of deep learning and probabilistic modeling, acting as a generative model that learns compact and meaningful representations of data in a latent, often lower-dimensional, space. At its core, VAE employs two primary components: an encoder and a decoder. The encoder's role is to process the input data, like an image or a text sequence, and convert it into a set of parameters defining a probability distribution in the latent space, commonly a Gaussian characterized by mean and variance. The decoder, on the other hand, takes a sample from this distribution and attempts to reconstruct the original input. The beauty of VAE's training process is its nuanced objective function: while one part aims to minimize the reconstruction error between the input and its VAE-produced counterpart, another imposes a regularization, pulling the learned latent distributions closer to a pre-defined one, typically a standard normal distribution. The objective of VAE is to maximize the ELBO:

$$\mathcal{L}(\theta, \phi; x^{(i)}) = \mathbb{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)] - D_{\mathrm{KL}}(q_\phi(z|x^{(i)})\|p(z)) \tag{2.3}$$

Where $x^{(i)}$ is a data point, z represents the latent variable, $p_\theta(x^{(i)}|z)$ is the probabilistic decoder, $p(z)$ stands as the prior on latent variables (typically a standard Gaussian distribution), $q_\phi(z|x^{(i)})$ is the probabilistic encoder or the approximate posterior, D_{KL} denotes the Kullback-Leibler divergence, and θ and ϕ are the parameters of the decoder and encoder, respectively. This careful balance ensures not just faithful data reconstructions but also a well-organized and continuous latent space, conducive for interpolation and generation tasks. The implications of VAE are profound. From generating entirely new images reminiscent of those in the training set, to filling in missing data, to serving as a foundation for more advanced models, its applications span across diverse domains, including but not limited to image synthesis, drug discovery, and anomaly detection.

2.5 Hierarchical Variational Autoencoder and Diffusion Model

Hierarchical Variational Autoencoders (HVAE) expand upon the foundational principles of traditional Variational Autoencoders (VAE). While VAEs aim to learn data distributions by encoding data into a latent space and then decoding it, HVAEs introduce multiple levels of latent hierarchies. This layered approach means that HVAEs don't rely on just a single latent representation; instead, they can have several, with each capturing different levels of abstraction within the data.

The philosophy behind HVAEs is somewhat analogous to our layered perception of abstract concepts. Just as we might interpret objects in our environment as manifestations of more

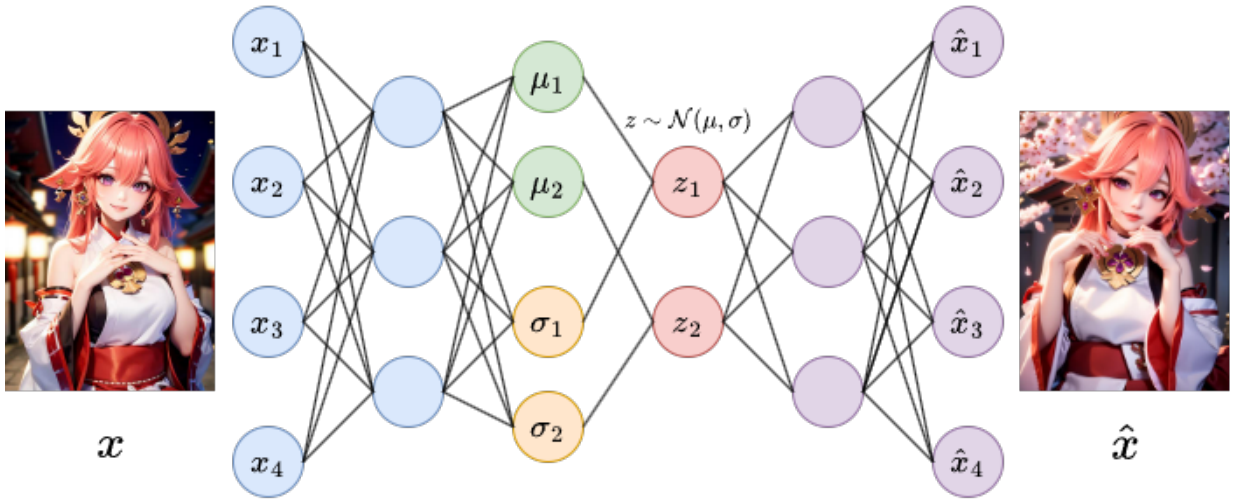

Figure 2.5: Structure of Variational Autoencoder.

abstract notions, HVAEs process data in a similar multi-layered manner. By incorporating multiple latent layers, the model can discern and represent more intricate patterns and relationships inherent in the data. A specific variant of HVAE, termed the Markovian HVAE (MHVAE), ensures that each latent layer is influenced only by the preceding one, creating a Markov chain of latents.

The multi-tiered structure of HVAEs offers a range of benefits. It facilitates the learning of richer and more nuanced data representations, proving invaluable for tasks demanding a profound understanding of data. The efficacy of models utilizing such hierarchical constructs highlights the potential and versatility of HVAEs in the realm of generative modeling. As the field advances, the multi-layered latent spaces of HVAEs are expected to spearhead the development of even more advanced and potent generative models.

Variational Diffusion Models (VDM) represent an innovative approach in generative modeling, combining features of Markovian Hierarchical Variational Autoencoders. The core idea behind VDM is to simulate the diffusion process of data across multiple timesteps, allowing for the generation of new data samples.

Three key characteristics define VDM:

1. The dimensionality of the latent space aligns perfectly with the dimensionality of the original data.

2. At each timestep, the structure of the latent encoder is predefined, rather than being learned. Specifically, it adopts a Gaussian distribution centered around the output from the previous timestep.

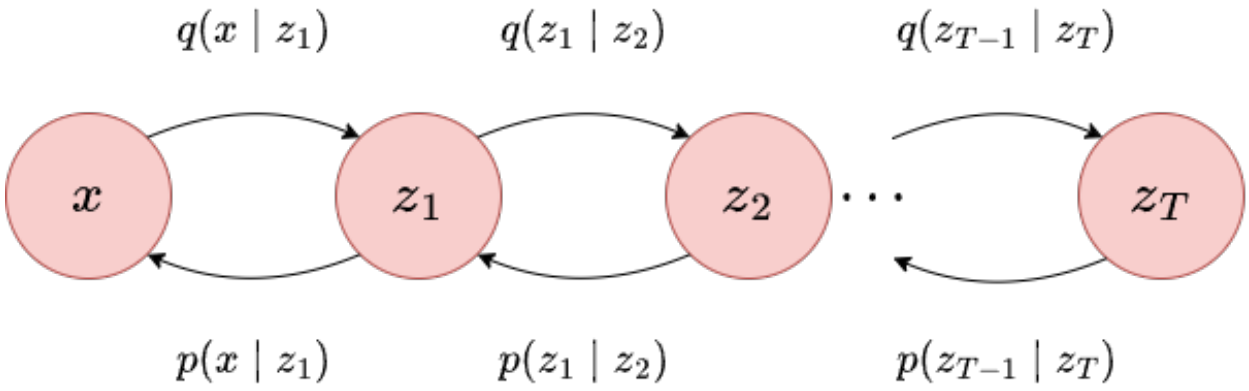

Figure 2.6: HVAE (Hierarchical Variational Autoencoders) is an extended generative model that introduces a hierarchical structure to capture complex structures in data, thereby enhancing the representation and generation capabilities of traditional VAEs.

3. The parameters of these Gaussian distributions evolve over time, ensuring that the latent distribution at the final timestep conforms to a standard Gaussian.

In essence, VDM models the diffusion process of data to generate new samples, enabling the capture of intricate patterns and relationships within the data. This offers a more flexible and potent method for data generation compared to traditional Variational Autoencoders.

In this chapter, we first introduced the Variational Autoencoder (VAE) and then explored its extended form, HVAE. We further delved into the discussion of diffusion models based on HVAE. This naturally raises a question: how many iterations are needed in the continuous process from x_0 to x_T for the diffusion model to produce high-quality images? In the following section, we will conduct an in-depth experimental exploration of this issue, using the classic DDPM diffusion model as an example.

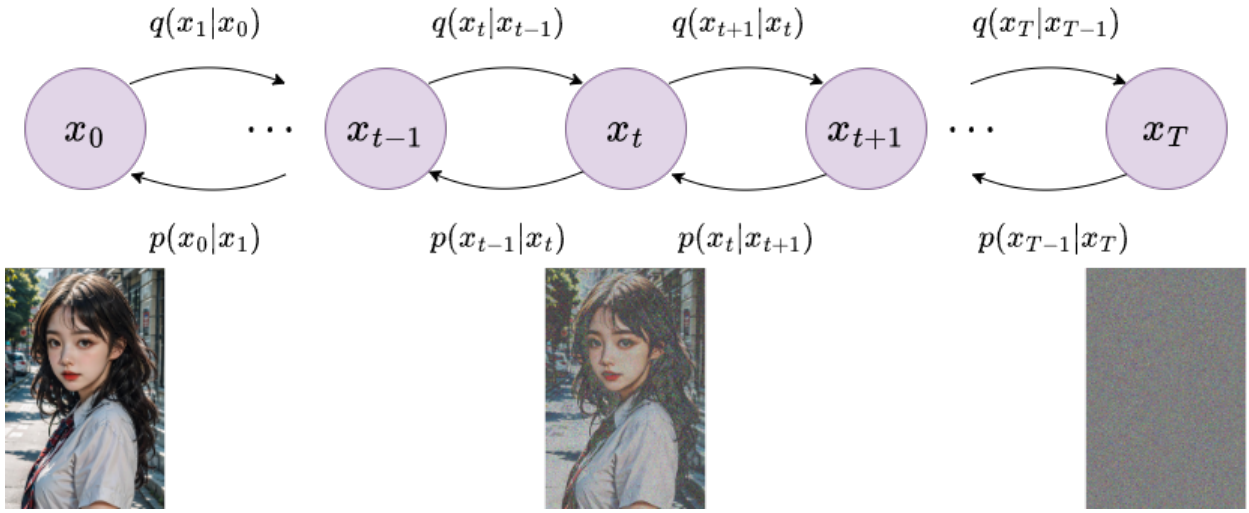

Figure 2.7: A Variational Diffusion Model (VDM) is a generative model that can be viewed as a Markovian Hierarchical Variational Autoencoder with specific constraints, combining Gaussian noise with data observations to generate data through a stepwise diffusion process.

Chapter 3

Methods

The diffusion model has emerged as the state-of-the-art generative model, producing image quality that significantly surpasses traditional GANs. Although the diffusion model was introduced in 2020 and has a brief history of just two to three years, it has already garnered widespread attention from researchers and has led to numerous scientific achievements. Currently, several renowned diffusion models exist, such as OpenAI's DALL-2 (Ramesh et al., 2022), Google's Imagen (Saharia et al., 2022), and beyond the academic realm, the widely acclaimed Stable Diffusion (Rombach et al., 2022) and Midjourney.

In this paper, our experiments employ the DDPM model (Ho, Jain, and Abbeel, 2020) which is based on non-equilibrium thermodynamics (Sohl-Dickstein et al., 2015) and score-based models (Song et al., 2021). These form the foundation for all diffusion models.

In this chapter, we will delve into the principles and specific experimental details involved in our research. In the subsequent chapter, we will present the results of the experiments described here. In Section 3.1, we will elucidate the fundamental principles of the diffusion model, including the forward diffusion process, denoise process, and training strategies. In Section 3.2, we will describe the neural network architecture we adopted. Section 3.3 will introduce the FID score, an indicator originally used to evaluate the image quality generated by GANs, but here, we use it to assess our trained diffusion model. In Section 3.4, we will detail the experimental methods of our study, encompassing the training strategy for the diffusion model and the approach to train classification neural networks using images generated by the diffusion model.

3.1 Diffusion Model

A diffusion (denoising) model appears relatively straightforward when juxtaposed with other generative models like Normalizing Flows, GANs, or VAEs. Essentially, all these models transform noise from a basic distribution into a data sample. Similarly, in this context, a neural network is trained to progressively refine data, initiating from sheer noise.

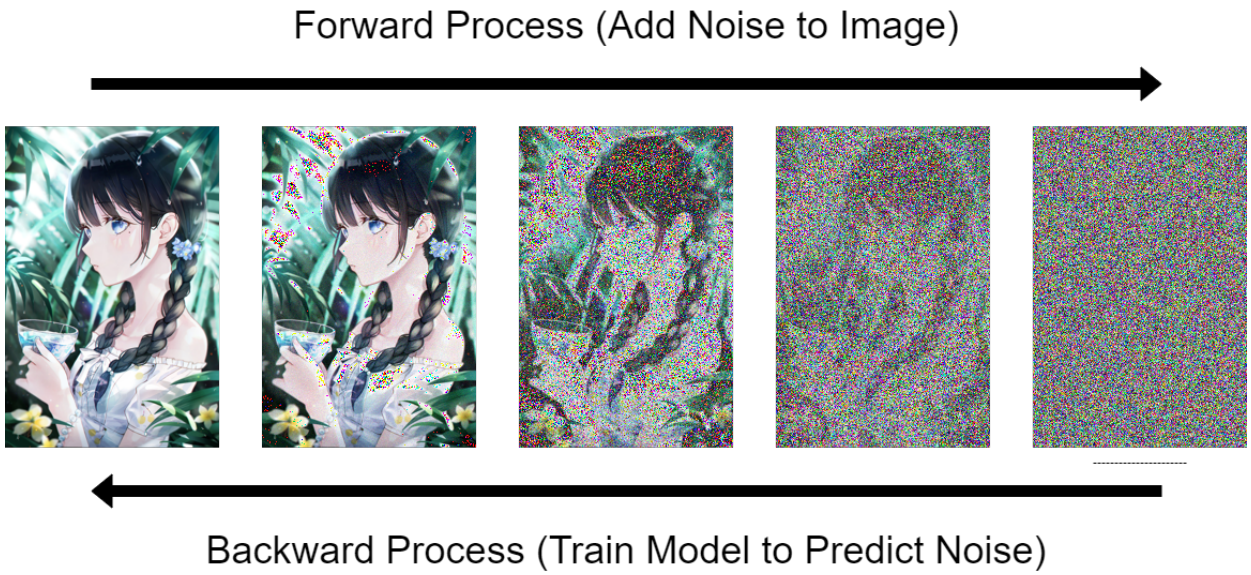

Figure 3.1: The diffusion model's process of adding noise to and denoising the training data.

Delving deeper into images, the architecture involves two distinct processes:

1. A predetermined forward diffusion process q that incrementally introduces Gaussian noise to an image until it becomes sheer noise.

2. A learned reverse denoising diffusion process p_θ, wherein a neural network is trained to methodically refine an image, starting from sheer noise, culminating in a genuine image.

Both the forward and reverse mechanisms, denoted by t, unfold over a set number of discrete intervals T. The process commences at $t = 0$, where an authentic image $\mathbf{x}_0$ is drawn from the dataset. During each interval t in the forward mechanism, noise, sourced from a Gaussian distribution, is amalgamated into the preceding image. With an adequately extensive T and a consistent strategy for noise introduction at every interval, the outcome is an isotropic Gaussian distribution at $t = T$, achieved through a phased approach.

3.1.1 Forward Process

To express this in a more structured manner, we ultimately require a comprehensible loss function for our neural network to optimize. Consider $q(\mathbf{x}_0)$ as the genuine data distribution, representing "authentic images". An image can be sampled from this distribution as $\mathbf{x}_0 \sim q(\mathbf{x}_0)$. We introduce the forward diffusion mechanism $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$, which incorporates Gaussian noise at each interval t, based on a predetermined variance sequence $0 < \beta_1 < \beta_2 < \ldots < \beta_T < 1$ given by:

$$q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)=\mathcal{N}\left(\mathbf{x}_t ; \sqrt{1-\beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \tag{3.1}$$

It's essential to remember that a Gaussian (or normal) distribution is characterized by two parameters: a mean μ and a non-negative variance σ^2. Essentially, each subsequent (marginally noisier) image at interval t is derived from a conditional Gaussian distribution with $\mu_t=\sqrt{1-\beta_t} \mathbf{x}_{t-1}$ and $\sigma_t^2=\beta_t$. This can be achieved by sampling $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then determining Equation (3.2).

$$\mathbf{x}_t=\sqrt{1-\beta_t} \mathbf{x}_{t-1}+\sqrt{\beta_t} \epsilon \tag{3.2}$$

It's noteworthy that the β_t values aren't consistent for each interval t (hence the subscript). One typically defines a "variance schedule", which might be linear, quadratic, cosine, etc., as we'll explore later, somewhat analogous to a learning rate schedule.

Starting with $\mathbf{x}_0$, we eventually obtain $\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T$, where $\mathbf{x}_T$ becomes pure Gaussian noise with an aptly set schedule. If we were aware of the conditional distribution $p\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)$, we could reverse the process: by drawing some arbitrary Gaussian noise $\mathbf{x}_T$, and then progressively "refining" it to eventually yield a sample from the genuine distribution $\mathbf{x}_0$.

We can compute the image with any number of added noise instances using an iterative method, as illustrated below:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{1-\beta_t} \mathbf{x}_{t-1}+\sqrt{\beta_t} \epsilon_{t-1} \\
\mathbf{x}_{t-1} &= \sqrt{1-\beta_{t-1}} \mathbf{x}_{t-2}+\sqrt{\beta_{t-1}} \epsilon_{t-2} \\
&\vdots \\
\mathbf{x}_1 &= \sqrt{1-\beta_1} \mathbf{x}_0+\sqrt{\beta_1} \epsilon_0
\end{aligned} \tag{3.3}$$

Define $\alpha_t:=1-\beta_t$ and $\bar{\alpha}_t:=\Pi_{s=1}^t \alpha_s$, and substituting the above equations from bottom to top, we can obtain:

$$\begin{aligned}
x_t &= \sqrt{1-\beta_t} \mathbf{x}_{t-1}+\sqrt{\beta_t} \epsilon_{t-1} \\
&= \sqrt{1-\beta_t}(\sqrt{1-\beta_{t-1}} \mathbf{x}_{t-2}+\sqrt{\beta_{t-1}} \epsilon_{t-2})+\sqrt{\beta_t} \epsilon_{t-1} \\
&= \sqrt{1-\beta_t} \sqrt{1-\beta_{t-1}} \mathbf{x}_{t-2}+\sqrt{1-\beta_t} \sqrt{\beta_{t-1}} \epsilon_{t-2}+\sqrt{\beta_t} \epsilon_{t-1} \\
&= \sqrt{\alpha_t} \sqrt{\alpha_{t-1}} \mathbf{x}_{t-2}+\sqrt{\alpha_t} \sqrt{1-\alpha_{t-1}} \epsilon_{t-2}+\sqrt{1-\alpha_t} \epsilon_{t-1} \\
&= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2}+\sqrt{\alpha_t-\alpha_t \alpha_{t-1}} \epsilon_{t-2}+\sqrt{1-\alpha_t} \epsilon_{t-1}
\end{aligned} \tag{3.4}$$

ϵ_{t-1} and ϵ_{t-1} in Equation (3.4) are two independent random variables that follow a Gaussian distribution. According to the sum of normally distributed random variables, we have:

$$\begin{aligned} x_t &= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{\sqrt{\alpha_t - \alpha_t \alpha_{t-1}}^2 + \sqrt{1-\alpha_t}^2} \epsilon_{t-2} \\ &= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \epsilon_{t-2} \end{aligned} \tag{3.5}$$

We sequentially substitute $x_{t-3}, x_{t-4}, \cdots, x_0$ into Equation (3.5), and then repeat this calculation process. Ultimately, we can obtain:

$$\begin{aligned} x_t &= \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t} \epsilon_0 \\ &\sim \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t} x_0, \left(1-\bar{\alpha}_t\right) \mathbf{I}\right) \end{aligned} \tag{3.6}$$

3.1.2 Denoise Process

In the limit as $T \to \infty$, the latent variable x_T converges to a standard isotropic Gaussian distribution. If we could deduce the inverse distribution $q\left(\mathbf{x}_{t-1} | \mathbf{x}_t\right)$, then by sampling x_T from $\mathcal{N}(0, \mathbf{I})$ and invoking the inverse operation, we could obtain a sample from $q\left(x_0\right)$, effectively replicating a data point from the initial data distribution. A central challenge is crafting a mechanism to simulate this backward diffusion.

Realistically, the exact form of $q\left(\mathbf{x}_{t-1} | \mathbf{x}_t\right)$ is not directly accessible. Direct computation is challenging, requiring intricate interactions with the primary data distribution.

We propose approximating $q\left(\mathbf{x}_{t-1} | \mathbf{x}_t\right)$ using a model p_θ, such as a neural network. Anticipating that $q\left(\mathbf{x}_{t-1} | \mathbf{x}_t\right)$ retains a Gaussian nature for small β_t, we model p_θ as Gaussian and modify its mean and variance:

$$p_\theta\left(\mathbf{x}_{t-1} | \mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right) \tag{3.7}$$

By systematically applying the inverse mechanism across all time points, represented as $p_\theta\left(\mathbf{x}_{0:T}\right)$ (commonly termed as the 'trajectory'), one can effectively transition from $\mathbf{x}_T$ back to the original data distribution. Mathematically, this is captured as:

$$p_\theta\left(\mathbf{x}_{0:T}\right) = p_\theta\left(\mathbf{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_{t-1} | \mathbf{x}_t\right) \tag{3.8}$$

By explicitly conditioning on each time point t, our model becomes adept at anticipating the Gaussian characteristics for each specific step. This entails the prediction of both the mean, $\boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right)$, and the covariance, $\boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)$.

3.1.3 Training

When training the diffusion model, we directed the model to learn the added noise rather than the images prior to noise addition. Consequently, the loss function for training the diffusion model is:

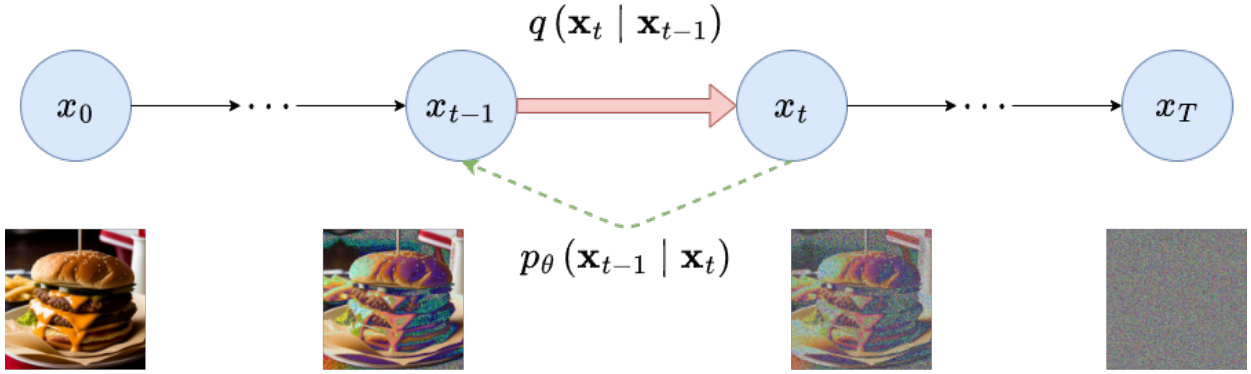

Figure 3.2: Deep diffusion models utilize a diffusion process to gradually add noise to data, and a denoising process to reverse this, transforming noise back into coherent data samples.

$$\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2 = \left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon, t\right)\right\|^2 \tag{3.9}$$

In the context, ϵ represents the actual added noise, while ϵ_θ signifies the deep neural network. This network takes two input parameters: the post-noise image $\mathbf{x}_t$ and the count t indicating how many times the image has been subjected to noise addition. The image $\mathbf{x}_t$ after noise addition can be articulated as $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$. Here, $\|\cdot\|^2$ denotes the squared Euclidean norm of a vector, employed to measure the vector's "length" or "magnitude".

To be specific, our training process can be delineated into the following steps:

- We obtain a random instance $\mathbf{x}_0$ from the genuine, potentially intricate, and unknown data distribution $q(\mathbf{x}_0)$.
- We determine a noise level t uniformly from an interval spanning 1 to T (essentially selecting a random temporal interval).
- We draw noise from a Gaussian distribution and modify the initial sample with this noise at the designated level t (capitalizing on the previously mentioned property).
- The neural model is optimized to estimate this noise using the perturbed image $\mathbf{x}_t$ (namely, noise introduced to $\mathbf{x}_0$ following the predetermined pattern β_t).

3.2 Network Architecture

Specifically, any deep neural network with consistent tensor shapes for both input and output can serve as a diffusion model, as the image post noise addition and the added noise itself share the same shape. From this perspective, most networks designed for semantic segmentation

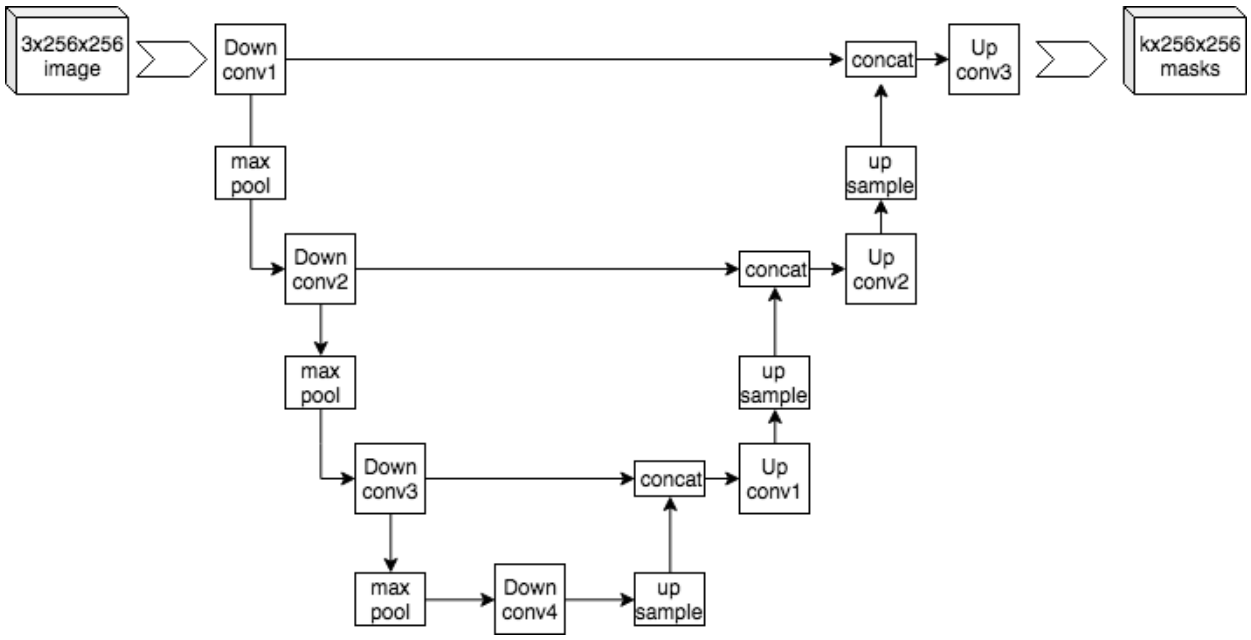

Figure 3.3: UCREL logo.

can be considered as diffusion models. In this study, we adopted the deep neural network used by DDPM, which is based on the U-Net architecture. U-Net is a classic neural network specifically designed for semantic segmentation, consisting of consecutive convolutional layers and residual connections. As an image is fed into this network, its dimensions gradually decrease while the channel count incrementally rises. Once the data passes through the network's central "bottleneck", the channel count starts to diminish, and the dimensions begin to expand, resulting in an output tensor shape identical to the input. Additionally, this network integrates group normalization (Y. Wu and He, 2018) and self attention (Vaswani et al., 2017) mechanisms.

Additionally, this neural network incorporates another scalar input, indicating the number of times noise has been added to the current input image. Upon entering the network, this information undergoes positional encoding and is integrated into the residual blocks.

3.3 Frechet Inception Distance (FID)

The Fréchet Inception Distance (FID) (Heusel et al., 2018) serves as a metric to evaluate the quality and diversity of synthetic images produced by Generative Adversarial Networks (GANs) and other generative models. As generative models have gained prominence in the realm of deep learning, there arose an immediate need for reliable evaluation metrics. Traditional measures, such as pixel-to-pixel differences, fall short as they fail to capture

perceptual and semantic differences between images. FID addresses this by comparing the statistical distribution of features extracted from real and generated images using a pretrained Inception model.

The Fréchet distance is a measure of similarity between two distributions. In the context of FID, the distributions of Inception features for real and generated images are considered. Specifically, given a set of real images X and generated images G, their feature vectors are extracted using the Inception network, yielding $f(X)$ and $f(G)$. These feature vectors can be viewed as being sampled from multivariate Gaussian distributions. Let m_X and m_G be the means of these distributions, and C_X and C_G be their covariance matrices. The Fréchet distance D^2 between these multivariate Gaussians is then computed as:

$$D^2(f(X), f(G)) = \|m_X - m_G\|^2 + \mathrm{Tr}(C_X + C_G - 2(C_X C_G)^{\frac{1}{2}})$$

This distance provides a scalar representation of the dissimilarity between the distributions of real and generated images in the feature space of the Inception network.

The reliance of FID on the feature space of the Inception network is crucial. The Inception model, having been trained on large and diverse datasets like ImageNet, captures a rich hierarchy of features ranging from simpler textures and edges at lower layers to complex object parts at deeper layers. By mapping images to this space, FID is comparing not the raw pixels but a higher-level representation of the images. Consequently, even a small FID value can indicate significant perceptual and semantic differences between sets of images. Moreover, unlike other metrics that capture only mode presence (how well patterns in real data are represented in generated data), FID captures both mode presence and mode dropping (whether the generative model produces artifacts not present in the real data). This makes it a more comprehensive measure for evaluating generative models.

In practice, FID has proven to be a robust metric that aligns well with human judgments of image quality. Its adoption has spread widely, and it's often used in tandem with other metrics, providing a holistic evaluation for generative models. However, a potential drawback is that FID is sensitive to the model used (in this case, Inception (Szegedy et al., 2016)) and may not be suitable for every dataset or generative task. As research progresses, it's crucial to consider the nuances and limitations of FID, but its current usage undoubtedly provides a consistent benchmark for assessing the performance of generative models across diverse challenges.

3.4 Experiment Detail

3.4.1 Program Environment

In this study, we purchased a Linux virtual machine provided by the public cloud service provider (Alibaba Cloud) and conducted all program experiments on this virtual machine. As

OS	Ubuntu 22.04
GPU	NVIDIA Tesla P40 / 24GB
CUDA	11.8
Python	3.8.17
PyTorch	2.0.1

Table 3.1: Experimental program's operating environment.

shown in Table 3.1, the operating system is Ubuntu 22.04. This virtual machine is equipped with an NVIDIA Tesla P40 graphics card. The experimental code was written in Python 3.8.17 and neural networks were built and trained using the PyTorch 2.0.1 framework.

3.4.2 Dataset

This study aims to explore the impact of different noise addition frequencies in diffusion models on the quality of generated images. To delve deeper into this, we decided to conduct multiple rounds of training on the same dataset using a uniform neural network structure. In these experiments, to ensure fairness in comparison, we fixed other hyperparameters (such as epoch size and batch size) and only adjusted the frequency of noise addition. However, relying solely on a single dataset might result in results being limited by the characteristics of that dataset, such as its size, texture, color, and structure of the images. To overcome this limitation, we selected three different datasets and adopted a consistent training method and result analysis, hoping to obtain more universally applicable conclusions.

Firstly, we trained using the Oxford Flower (Nilsback and Zisserman, 2008) dataset. This dataset focuses on image classification, covering 102 common flowers in the UK, with a total of 8189 images, with at least 40 images for each flower type. Unlike other classification datasets, the fine-grained task of Oxford Flower challenges the model's ability to distinguish flowers that look similar but belong to different categories. In the training of the diffusion model, we used 7169 training set images, expecting the model to generate synthetic images with specific flower features or a fusion of features from multiple flowers.

Next, we used the Smithsonian Butterfly dataset provided by Hugging Face (Cinarel, 2023). This dataset focuses on pictures of different types of butterflies. There are three reasons for choosing this dataset: Firstly, its moderate size, containing 1000 images, ensures that the model learns rich image features while avoiding the long training time required by large-scale datasets; Secondly, the butterfly in each image almost occupies the entire image space, which helps the model focus more on learning the image features of the butterfly; Lastly, all images have a pure white background, which helps the model reduce learning from non-target information.

Lastly, we used the Anime Face dataset from Kaggle (An, 2021), which is a dataset containing nearly 300,000 Japanese anime avatars. For training, we selected 14,000 images

Figure 3.4: 8 image samples from the Oxford Flower dataset.

from it as training samples.

3.4.3 Preprocessing

Before starting the training, we preprocessed the data. First, all training images were resized to 128x128 while retaining the RGB color channels and removing the alpha transparency channel. Next, we applied a simple augmentation to the dataset, randomly selecting half of the images for horizontal flipping. Finally, the tensor values were adjusted from the range [0,255] to [0,1], and each channel was normalized as shown in Equation (3.10).

$$\text{channel} = \frac{\text{channel} - \text{mean}}{\text{std}} \tag{3.10}$$

In this equation, channel is a specific image channel, mean represents the average value of the channel, and std is the standard deviation of the channel. In the experiments conducted in this paper, both the mean and variance were chosen to be 0.5. This normalization helps in stabilizing the training process and achieving faster convergence in deep learning models.

3.4.4 Training

After the data preprocessing was completed, we began training the neural network. The neural network learned to predict the added noise, and we trained for a total of 50 epochs with a batch size set to 16. We used the Adam optimizer to update the gradients of the neural network and set the learning rate of the Adam optimizer to 1×10^{-4}. We employed a cosine annealing mechanism (Loshchilov and Hutter, 2017) to update the optimizer's learning rate. This mechanism is divided into two phases, with the first phase as shown in Equation (3.11).

Figure 3.5: 8 image samples from the Smithsonian Butterfly dataset.

Figure 3.6: 8 image samples from the Anime Face dataset.

$$\text{lr}(t) = \text{lr}_{\text{base}} \times \frac{t}{\text{warmup\_steps}} \tag{3.11}$$

The function $\text{lr}(t)$ represents the learning rate at time t, which is the current training step or iteration. It's a variable that changes over time, indicating the learning rate to be used at a specific training step. The term lr_{base} is the predetermined initial learning rate, typically the learning rate you set before starting the training. At the end of the warmup phase, the learning rate will reach this value. Then, in the cosine annealing phase, the learning rate will start decreasing from this value. The variable t stands for the current training step or iteration. As t starts from 0 and gradually increases, the learning rate also starts from 0 and increases linearly until it reaches lr_{base}. The term warmup_steps represents the total number of steps or iterations for the warmup phase. It defines the duration of the warmup phase. Once this phase is over, the learning rate will have reached lr_{base}, and then the cosine annealing phase will begin.

The formula for the cosine annealing phase is given by:

$$\text{lr}(t) = \text{lr}_{\text{min}} + \frac{1}{2}(\text{lr}_{\text{base}} - \text{lr}_{\text{min}})(1 + \cos(\frac{t - \text{warmup\_steps}}{T - \text{warmup\_steps}}\pi)) \tag{3.12}$$

where $\text{lr}(t)$ represents the learning rate at time t, the current training step or iteration after the warmup phase. It's a variable that changes over time, indicating the learning rate to be used at a specific training step. The term lr_{base} is the predetermined initial learning rate, typically set before starting the training, and serves as the maximum learning rate used after the warmup phase. lr_{min} is the minimum learning rate to which the scheduler will anneal, ensuring that as training progresses, the learning rate decreases but doesn't reach zero. The variable t stands for the current training step or iteration and determines the current position within the cosine curve. warmup_steps represents the total number of steps or iterations for the warmup phase and offsets t so that the cosine annealing starts right after the warmup. T denotes the total number of training steps or iterations and normalizes the cosine function over the training duration. The cosine function, $\cos(\cdot)$, gives the annealing phase its name and shape, ensuring a smooth, cyclical decrease in the learning rate, resembling half a period of the cosine curve.

In our experiments, we conducted 20 rounds of training, adjusting only the number of times noise was added in each round while keeping all other parameters consistent. Initially, we debugged the model's code on a Jupyter notebook using a subset of training images. Once verified, we transitioned it into a standalone Python script and proceeded with its engineering aspects.

In our engineering practices:

- We set the training process as a Linux background task and utilized the open-source `supervisor` process manager for monitoring on the server.

- Opting for efficiency in logging, we replaced the command-line progress display, `tqdm`, with Python's `logging` module, directing logs to files. These logs are updated daily, named after the current date. After every four training steps, the current progress is logged.
- Additionally, we integrated `TensorBoard`, a web server allowing real-time training monitoring via a browser.
- We also employed the public cloud service `wandb` for tracking neural network training. The program periodically pushes updates to the `wandb` server. In events like training completion or process crashes, `wandb` sends notifications via Slack and email.
- Given the extended training duration, to prevent restarts due to unforeseen interruptions, we adopted a strategy of saving the current model every five epochs and simultaneously deleting the previous one. This rolling update approach ensures quick resumption from the last checkpoint in case of crashes.
- To enhance training efficiency, we employed mixed-precision training using both 16-bit (half-precision or FP16) and 32-bit (single-precision or FP32) floating-point numbers, reducing GPU memory consumption.
- Lastly, for streamlined training management, we utilized the `accelerate` library from Hugging Face, which is highly compatible with native PyTorch code.

3.4.5 Analyzing

Upon completing the training for each dataset, we obtained 20 checkpoints, each corresponding to a different number of noise additions. We had each of these 20 models generate images. Theoretically, the more images generated, the better the model's quality can be assessed. However, due to hardware limitations in our experiments, we opted for each model to produce 128 images. While diffusion models, in theory, can generate any number of images simultaneously, the more images generated at once, the more GPU memory is required. After several trials, we decided to have the diffusion model generate 8 images in parallel, repeating this process 16 times. Once all 128 images were produced, we computed the FID value using the images generated by the algorithm described in Section 3.3 and those from the diffusion model training.

3.4.6 Classification

After the completion of the diffusion model training, we utilized the original images from the training set and the images generated by the diffusion model to train the classification neural network. This attempt aims to use the images produced by the diffusion model as a

data augmentation technique. Traditional data augmentation methods (Yang et al., 2022) are mostly based on rotating, translating, and flipping a single image from the training set. In contrast, our new strategy directly employs the diffusion model to generate images, thereby expanding the training set's size.

Specifically, we constructed three datasets:

1. Images from the Oxford Flower and Smithsonian Butterfly training sets, 150 images each, named "Origin".

2. Images generated using the diffusion models trained on Oxford Flower and Smithsonian Butterfly, 150 images each, named "Synthesis".

3. A training set formed by mixing the first two datasets, named "Mix".

All three training sets use images from the Oxford Flower and Smithsonian Butterfly test sets as their test images, implying that the images in the test set were not used for training the diffusion model.

Image classification is a common task in deep learning within the field of computer vision, and many mature and efficient neural network structures have been developed. We chose two classic image classification neural networks, ResNet18 (He et al., 2016) and VGG16 (Simonyan and Zisserman, 2014), to train these three datasets and observed the changes in accuracy. Instead of designing a neural network from scratch, we used the networks pre-trained on ImageNet from PyTorch's torchvision. Subsequently, we initialized these networks' weights to random values for a fresh start in training.

Chapter 4

Results

We trained the diffusion model on three distinct datasets. For each dataset, we conducted 20 rounds of training, with each round adopting a different number of noise addition instances. As a result, we obtained a total of 60 individual models. The average time required to train the diffusion model on each dataset is listed in Table 4.1.

Thus, our total training time was $(15 + 2 + 50) \times 20 = 940$ hours.

In this context, the mentioned time refers to the overall training duration. In our experiment, we implemented a parallel training strategy. Initially, we successfully set up CUDA, Anaconda, Python, PyTorch, and other related third-party libraries on a server equipped with an NVIDIA graphics card. After debugging the code on a small dataset and ensuring its successful execution, we cloned the complete operating system of the server. Subsequently, we configured different parameters on multiple virtual machines and initiated the training process simultaneously.

Figure 4.1, Figure 4.2, and Figure 4.3 display images generated by the diffusion model. Each row showcases images produced by the same diffusion model, totaling ten rows, which represent the number of noise additions ranging from 100 to 1000, in ten increments. Due to space constraints on the page, we haven't shown the scenarios for noise additions ranging from 1100 to 2000. Upon observation, images generated with noise additions beyond 1000 closely resemble those produced with 1000 noise additions. Further details can be referred to in the subsequent tables presenting the FID values. From these three figures, it's evident that the quality of images generated by the diffusion model significantly improves initially as the

Dataset	Training Time
Oxford Flower	15 hours
Smithsonian Butterfly	2 hours
Anime Face	30 hours

Table 4.1: The average time spent training a diffusion model for each dataset.

Inference Step	FID Score		
	Oxford Flower	Smithsonian Butterfly	Anime Face
100	369.88	464.27	313.47
200	208.77	414.54	270.74
300	176.46	436.99	229.2
400	150.36	371.72	119.72
500	125.77	349.58	100.96
600	116.52	325.88	95.55
700	117.46	316.48	96.32
800	122.95	317.76	96.5
900	133.55	314.49	109.64
1000	142.56	313.22	114.22
1100	146.5	319.56	114.29
1200	141.07	316.08	115.73
1300	127.79	314.74	107.03
1400	140.6	314.38	117.4
1500	132.09	314.96	111.05
1600	145.1	315.84	113.85
1700	139.23	308.78	117.6
1800	134.86	315.87	111.06
1900	139.89	309.66	118.8
2000	144.07	321.56	109.04

Table 4.2: On three different datasets, we used varying numbers of noise additions to calculate the FID score for the diffusion model.

number of noise additions increases. With fewer additions, the content of the generated images is almost indistinguishable. However, as the number grows, the images become progressively clearer and of higher quality. Eventually, as the number of noise additions continues to rise, the quality of the generated images starts to plateau.

Table 4.2 fully displays the FID values of images generated by different diffusion models. Figure 4.4, Figure 4.5, and Figure 4.6 show the trend of FID values for a single dataset under different numbers of noise additions.

Table 4.3 and Figure 4.7 illustrate the time taken by the diffusion model to generate 128 images, each of size 128x128. We only present a single table here without specifying the datasets. This is because, despite the differences in the size of the training sets for these three datasets, they consumed identical time for image generation due to the utilization of the same neural network architecture and preprocessing procedures.

Table 1 displays the accuracy of ResNet18 and VGG16 on the three datasets described in

Figure 4.1: Images generated by the diffusion model trained on the Oxford Flower dataset.

Figure 4.2: Images generated by the diffusion model trained on the Smithsonian Butterfly dataset.

Figure 4.3: Images generated by the diffusion model trained on the Anime Face dataset.

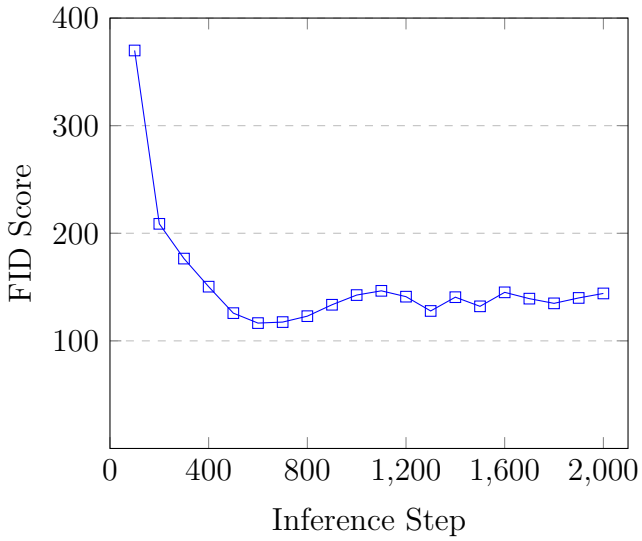

Figure 4.4: For the Oxford Flower dataset, as the number of noise additions increased, there was a change in the FID score of the images generated by the diffusion model.

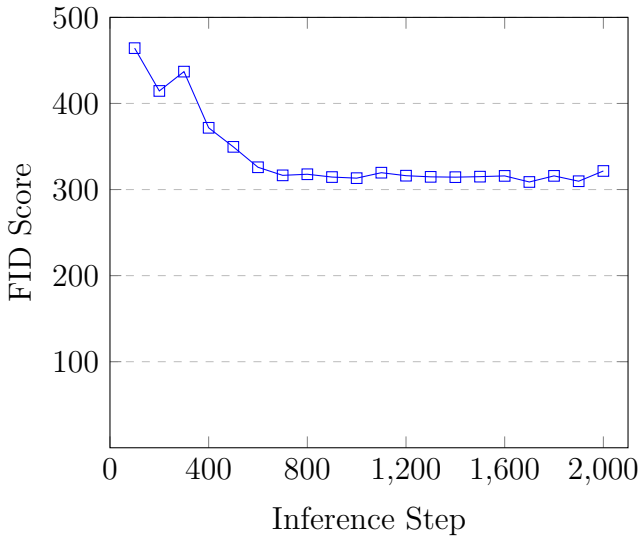

Figure 4.5: For the Smithsonian Butterfly dataset, as the number of noise additions increased, there was a change in the FID score of the images generated by the diffusion model.

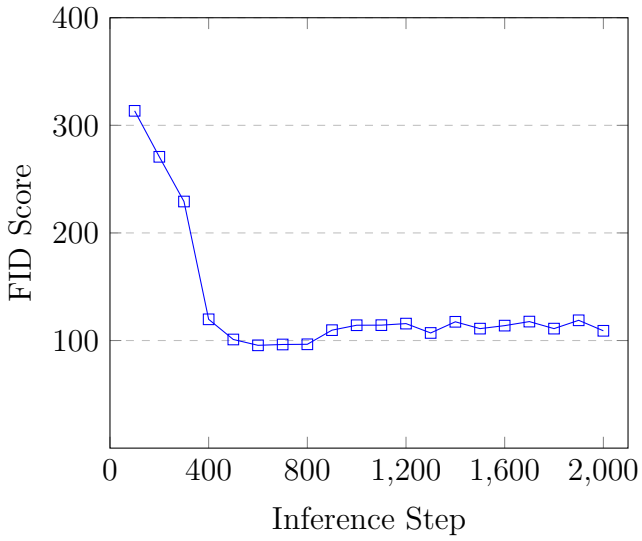

Figure 4.6: For the Anime Face dataset, as the number of noise additions increased, there was a change in the FID score of the images generated by the diffusion model.

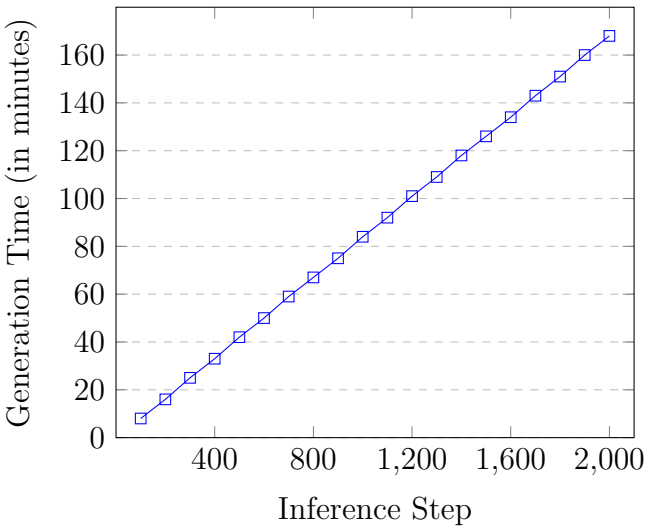

Figure 4.7: The time taken by the diffusion model to generate 128 images of size 128x128.

Inference Step	Generation Time
100	0:08:20
200	0:16:50
300	0:25:10
400	0:33:40
500	0:42:10
600	0:50:30
700	0:59:00
800	1:07:20
900	1:15:50
1000	1:24:20
1100	1:32:40
1200	1:41:10
1300	1:49:30
1400	1:58:00
1500	2:06:20
1600	2:14:40
1700	2:23:10
1800	2:31:40
1900	2:40:10
2000	2:48:40

Table 4.3: The time taken by the diffusion model to generate 128 images of size 128x128.

	Origin	Synthesis	Mix
ResNet18	65%	59%	(65+7)=72%
VGG16	61%	54%	(61+9)=70%

Table 4.4: The accuracy of using ResNet18 and VGG16 on three datasets.

the previous section. From the table, we can observe that even when training deep neural networks using images generated by the diffusion model, we can achieve reasonably good results, though the accuracy is still slightly lower than when training directly with the original dataset. Looking at the last column of the table, we notice that compared to training solely on the original dataset, the accuracy of the classification model trained with both the original and diffusion model-generated data shows further improvement.

Chapter 5

Discussion

Through the experimental process described in Chapter 3 and the results presented in Chapter 4, it becomes evident that the number of noise additions in diffusion models significantly impacts the quality of image generation. However, the degree of this impact varies at different stages. In the initial stages, when the noise additions are relatively few, the effect on the image quality generated by the diffusion model is quite pronounced. This is evident from the illustrations in Chapter 2, where the quality of images generated by the diffusion model progressively improves.

As the number of noise additions increases, there comes a point where the quality of the images generated by the diffusion model plateaus and no longer changes. In the previous chapter, we trained numerous models with varying noise addition frequencies (a total of 60, with each dataset trained 20 times, and noise additions ranging from 100 to 2000). We then used these models to generate 128 images each. Observing these images, especially with fewer noise additions like 100 or 200 times, the images generated by the diffusion model were almost unrecognizable. At this stage, the images generated by the diffusion model were essentially indistinguishable and resembled Gaussian noise. Fewer sampling steps resulted in poorer image quality. We believe the primary reason for this is that during the series of noise additions, the proportion of noise added each time was so significant that it masked the inherent texture, color, and structural features of the training images, which are precisely what the diffusion model needs to learn during training.

Theoretically speaking, the starting and ending points of this series of noise additions in the diffusion model are fixed. The starting point is the images in the training set, and the endpoint is an image that follows a multivariate Gaussian distribution. To draw an analogy, the journey from the original training set image to the Gaussian noise is a fixed distance. The series of noise additions breaks this journey into multiple segments. Fewer noise additions mean the model has to learn more information each time, exceeding the capacity of the neural network of the diffusion model. This can lead to the loss of some valuable information.

Furthermore, we observed that although the quality of images generated by the diffusion

model improves with an increasing number of noise additions, this trend doesn't continue indefinitely. In the experiments of the previous chapter, under consistent software and hardware conditions and with all training hyperparameters kept constant, we found that once the number of noise additions reached a certain threshold (close to 1000 times), the quality of the images generated by the trained diffusion model no longer improved. However, the image generation time continued to grow proportionally with the number of noise additions. Thus, to achieve higher image quality, other optimizations are necessary, such as using larger-scale deep neural networks. The development of the GPT series models in the NLP domain has clearly demonstrated the positive impact of enlarging the parameter scale of deep neural networks.

In this experiment, we used a UNet neural network with a total of 113,673,219 trainable parameters. Due to hardware limitations (mainly the graphics card), we found it challenging to train larger models. We have reason to believe that larger models might produce stable-quality images with fewer noise additions. Additionally, we also explored using images generated by the diffusion model to augment the training set for image classification neural networks, which can enhance the accuracy of image classification networks on the test set. We believe this approach can serve as a novel image enhancement method. Traditional data augmentation methods focus on individual images, performing operations like flipping, translating, and rotating. Such methods have their limitations because, although they increase the number of training images, the main information of an image remains unchanged. Training data with diffusion models and then generating images can increase dataset diversity. Most importantly, while traditional image augmentation methods can expand the dataset size, they have a limit. In contrast, generating images with diffusion models can infinitely expand the training set size. We believe this method can be effective in scenarios where a large number of similar images are needed. In such cases, one can first train a diffusion model and then use it to generate a large number of similar images.

Chapter 6

Conclusions

In this study, we delved deeply into generative models in the field of deep learning. Firstly, we systematically reviewed the development trends of deep generative models in recent years, especially mainstream models such as VAE and GAN. We detailed their working principles, various improvement strategies, and their respective advantages and limitations. Notably, although these models differ in characteristics, they all share a core premise: the training data comes from an unknown probability distribution. To uncover this hidden distribution and generate new data from it, these models all attempt to construct a mathematical model based on probability. Mathematically, the real challenge is how to estimate this unknown probability model based on a limited number of samples.

In this process, all these models rely on a specific function, the computation of which is quite complex. Even with modern high-performance computers, it's challenging to determine an exact algorithm to obtain it. Fortunately, due to the excellent function approximation ability of deep neural networks, they can mathematically fit any function, providing us with an effective solution. Furthermore, we deeply analyzed several mainstream generative models and discussed their basic principles, advantages, and limitations. Among them, the diffusion model is particularly noteworthy, with its design concept originating from non-equilibrium thermodynamics in physics.

Although the theoretical foundation of the diffusion model was proposed as early as 2015, it was once surpassed by GAN in terms of development speed. Compared to the direct and concise structure of GAN, the diffusion model appears more complex. However, by 2020, thanks to more powerful graphics cards and larger datasets, researchers successfully combined the diffusion model with deep neural networks, creating DDPM, and achieved results in image generation that surpassed GAN. With the emergence of breakthrough research like latent diffusion, the diffusion model quickly rose to prominence. Currently, most high-quality AI-generated images, such as those produced by Stable Diffusion and Midjourney, are on par with ChatGPT and are considered leaders in the AIGC field.

However, although the diffusion model surpasses GAN in image quality and diversity, its

generation speed is relatively slow, and it requires more computational resources. For example, using an NVIDIA RTX 4090 graphics card, Stable Diffusion might take tens of seconds to generate an image, while GAN is faster. This is mainly because the diffusion model needs to perform multiple forward diffusions when generating an image, and due to its relatively large neural network structure, each forward diffusion requires a lot of computational resources. To address this issue, we specifically studied the relationship between the number of noise additions and image generation quality. Lastly, we found that combining images generated by the diffusion model with images from the original training set for training a classification network can significantly improve the model's performance. This means that using images generated by the diffusion model as training data may be more effective than traditional image augmentation methods. Research in this area is still in its early stages, but this issue undoubtedly has significant research value.

References

An, Subin (2021). *High-Resolution Anime Face Dataset (512x512)*. URL: https://www.kaggle.com/datasets/subinium/highresolution-anime-face-dataset-512x512 (visited on 09/05/2023).

Brown, Tom et al. (2020). "Language Models Are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html (visited on 09/05/2023).

Cho, Kyunghyun et al. (Oct. 2014). "Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: https://aclanthology.org/D14-1179 (visited on 08/27/2023).

Cinarel, Ceyda (Apr. 2023). *Ceyda/Smithsonian_butterflies · Datasets at Hugging Face*. URL: https://huggingface.co/datasets/ceyda/smithsonian_butterflies (visited on 09/05/2023).

Deng, Jia et al. (June 2009). "ImageNet: A Large-Scale Hierarchical Image Database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

Goodfellow, Ian et al. (2014). "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

He, Kaiming et al. (June 2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 08/26/2023).

Heusel, Martin et al. (Jan. 2018). *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. DOI: 10.48550/arXiv.1706.08500. arXiv: 1706.08500 [cs, stat]. URL: http://arxiv.org/abs/1706.08500 (visited on 09/05/2023).

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates,

Inc., pp. 6840–6851. URL: https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html (visited on 08/20/2023).

Kingma, Diederik P. and Max Welling (Dec. 2013). *Auto-Encoding Variational Bayes*. DOI: 10.48550/arXiv.1312.6114. arXiv: 1312.6114 [cs, stat]. URL: http://arxiv.org/abs/1312.6114 (visited on 09/05/2023).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 08/20/2023).

Loshchilov, Ilya and Frank Hutter (May 2017). *SGDR: Stochastic Gradient Descent with Warm Restarts*. DOI: 10.48550/arXiv.1608.03983. arXiv: 1608.03983 [cs, math]. URL: http://arxiv.org/abs/1608.03983 (visited on 09/06/2023).

Luo, Calvin (Aug. 2022). *Understanding Diffusion Models: A Unified Perspective*. arXiv: 2208.11970 [cs]. URL: http://arxiv.org/abs/2208.11970 (visited on 08/26/2023).

Nilsback, Maria-Elena and Andrew Zisserman (Dec. 2008). "Automated Flower Classification over a Large Number of Classes". In: *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. Bhubaneswar, India: IEEE, pp. 722–729. DOI: 10.1109/ICVGIP.2008.47. URL: http://ieeexplore.ieee.org/document/4756141/ (visited on 08/19/2023).

Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu (Aug. 2016). *Pixel Recurrent Neural Networks*. DOI: 10.48550/arXiv.1601.06759. arXiv: 1601.06759 [cs]. URL: http://arxiv.org/abs/1601.06759 (visited on 08/27/2023).

OpenAI (Mar. 2023a). *GPT-4 Technical Report*. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs]. URL: http://arxiv.org/abs/2303.08774 (visited on 09/05/2023).

— (2023b). *Generative Models*. URL: https://openai.com/research/generative-models (visited on 08/31/2023).

Radford, Alec, Karthik Narasimhan, et al. (2018). "Improving Language Understanding by Generative Pre-Training". In.

Radford, Alec, Jeffrey Wu, et al. (2019). "Language Models Are Unsupervised Multitask Learners". In.

Ramesh, Aditya et al. (Apr. 2022). *Hierarchical Text-Conditional Image Generation with CLIP Latents*. DOI: 10.48550/arXiv.2204.06125. arXiv: 2204.06125 [cs]. URL: http://arxiv.org/abs/2204.06125 (visited on 08/27/2023).

Rombach, Robin et al. (June 2022). "High-Resolution Image Synthesis with Latent Diffusion Models". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, pp. 10674–10685. ISBN: 978-1-66546-946-3. DOI: 10.1109/CVPR52688.2022.01042. URL: https://ieeexplore.ieee.org/document/9878449/ (visited on 08/27/2023).

Saharia, Chitwan et al. (May 2022). *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. DOI: 10.48550/arXiv.2205.11487. arXiv: 2205.11487 [cs]. URL: http://arxiv.org/abs/2205.11487 (visited on 08/27/2023).

Salimans, Tim et al. (Jan. 2017). *PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications*. DOI: 10.48550/arXiv.1701.05517. arXiv: 1701.05517 [cs, stat]. URL: http://arxiv.org/abs/1701.05517 (visited on 08/27/2023).

Simonyan, Karen and Andrew Zisserman (Sept. 2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. URL: https://arxiv.org/abs/1409.1556v6 (visited on 09/06/2023).

Sohl-Dickstein, Jascha et al. (June 2015). "Deep Unsupervised Learning Using Nonequilibrium Thermodynamics". In: *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, pp. 2256–2265. URL: https://proceedings.mlr.press/v37/sohl-dickstein15.html (visited on 09/05/2023).

Song, Yang et al. (Feb. 2021). *Score-Based Generative Modeling through Stochastic Differential Equations*. DOI: 10.48550/arXiv.2011.13456. arXiv: 2011.13456 [cs, stat]. URL: http://arxiv.org/abs/2011.13456 (visited on 09/05/2023).

Szegedy, Christian et al. (2016). "Rethinking the Inception Architecture for Computer Vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.html (visited on 09/05/2023).

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wu, Yuxin and Kaiming He (June 2018). *Group Normalization*. DOI: 10.48550/arXiv.1803.08494. arXiv: 1803.08494 [cs]. URL: http://arxiv.org/abs/1803.08494 (visited on 09/08/2023).

Yang, Suorong et al. (Apr. 2022). *Image Data Augmentation for Deep Learning: A Survey*. DOI: 10.48550/arXiv.2204.08610. arXiv: 2204.08610 [cs]. URL: http://arxiv.org/abs/2204.08610 (visited on 09/06/2023).