DiffusionCNF: Learning Denoising Diffusion Models via Conditional Normalizing Flows

by

Sanika Saurabh Katekar

(Under the Direction of Jaewoo Lee)

Abstract

In recent years, denoising diffusion models have demonstrated remarkable performance in the realm of generative modeling. However, these models have a significant drawback which is that of slow sampling time. The main cause of their high sampling cost is due to the Gaussian assumption of the reverse Markov transition, which requires a large number of denoising steps. To address this issue, we present a novel method, called DiffusionCNF, which models the reverse process of the diffusion model using normalizing flows. Our proposed approach enhances the sampling speed while maintaining the desirable properties of diffusion models. By leveraging the strengths of both models, we contribute to advancing the field of generative modeling and offer a promising solution for efficient and effective generation of complex data distributions.

Index words: Generative Model, Diffusion Model, Normalizing Flow, DiffusionCNF

DiffusionCNF: Learning Denoising Diffusion Models via Conditional Normalizing Flows

by

Sanika Saurabh Katekar

B.Tech., MIT World Peace University, India, 2021

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

MASTER OF SCIENCE

Athens, Georgia

2023

DiffusionCNF: Learning Denoising Diffusion Models via Conditional Normalizing Flows

by

Sanika Saurabh Katekar

Acknowledgments

I am immensely grateful to Professor Jaewoo Lee, who has been an exceptional mentor, for his invaluable guidance and unwavering support throughout my graduate studies. His profound expertise and assistance have played a crucial role in molding my research and propelling me towards the accomplishment of my academic ambitions. Additionally, I would like to express my gratitude to Professor Frederick Maier and Professor Ninghao Liu, distinguished members of my committee, for their valuable input that significantly elevated the caliber of my thesis. Their insightful perspectives and profound expertise have broadened my comprehension of the subject matter and ignited my curiosity to explore uncharted research directions. Furthermore, I wish to acknowledge the significant contributions made by my fellow MSAI classmate, Soham Sajekar. His valuable insights and collaborative efforts have undeniably enhanced the richness of this thesis. I want to thank University of Georgia for giving me a stimulating academic setting and lots of chances for professional growth. The college's resources and assistance have been crucial in assisting me to finish my thesis. Lastly, I want to express my deepest gratitude to my cherished family members for their support and encouragement throughout my academic journey. Their presence and belief in my abilities have served as a constant source of resilience and motivation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Generative AI models have emerged as effective tools for producing diverse and innovative samples across various data types, including text [3, 5], images [37, 22, 36, 8], and videos [20, 2]. These models leverage complex data distributions to unlock creative possibilities in computer vision and natural language processing. In the field of computer vision, generative models have been applied successfully to generate realistic images of objects, faces, and complex scenes [21, 37]. By training on extensive datasets, these models acquire the ability to generate new data that closely resembles the patterns and structures observed in the training data. This capability has proven valuable in tasks such as image synthesis [34, 8, 36], data augmentation, and even artistic creation [37]. Similarly, in natural language processing, generative models have been utilized to develop conversational agents and chatbots with human-like language skills [5]. Through training on large text corpora, these models learn the underlying structures and semantic relationships present in the data. As a result, they can produce coherent and contextually appropriate responses, leading to engaging and interactive conversational experiences.

Among the various types of generative models, one particularly prominent architecture is the Generative Adversarial Network (GAN) [16, 22, 15, 35]. GANs consist of two interconnected components: a generator and a discriminator. The generator's role is to produce synthetic data, such as images [7] or text [14], while the discriminator's task is to distinguish between real and fake samples. Through an iterative process of competition and feedback, the generator learns to produce increasingly realistic and convincing outputs, while the discriminator improves its ability to differentiate between real and generated data. Despite their remarkable capabilities, GANs do have some limitations. One challenge is their inherent instability during training, often manifesting as mode collapse [30]. This issue can make training GANs a delicate and time-consuming process, requiring careful parameter tuning and architectural modifications. To address some of the challenges of GANs, researchers have explored alternative generative models, such as Denoising Diffusion Probabilistic Models (DDPM) [21] and Normalizing Flows [10].

Denoising Diffusion Probabilistic Models (DDPM) [21] represent a promising alternative to address the challenges faced by Generative Adversarial Networks (GANs). DDPMs leverage the concept of diffusion models [41], which operate through a two-step process known as the forward process and the reverse process. In the forward process of DDPMs, Gaussian noise is gradually added to the training data, resulting in a sequence of images that become progressively more distorted. This process transforms the original data into a distribution that is entirely Gaussian. On the other hand, the reverse process involves the utilization of a neural network to denoise the image obtained from the forward process. By applying the reverse network in a step-by-step fashion, the Gaussian noise added during the forward process is effectively removed, restoring the image to its original form. It is important to note that the forward process in DDPM is based on a Markov Chain process, where each step introduces a specific amount of noise. The neural network in the reverse process is trained to learn how to denoise the noisy samples. The initial diffusion model introduced in the literature [41] was not renowned for its ability to produce high-quality samples. However, the introduction of DDPM [21] significantly improved the performance of diffusion models in terms of sample quality. The authors of DDPM successfully constructed a diffusion model capable of generating samples of remarkable fidelity and realism. Moreover, the authors of DDPM suggested that a specific parameterization of diffusion models allows them to be considered equivalent to other approaches in generative modeling and sampling, such as denoising score matching and annealed Langevin dynamics. This intriguing connection broadens the understanding of diffusion models.

While Denoising Diffusion Probabilistic Models (DDPMs), along with further literature that provided more advancement, have made significant strides in improving sample quality, note that the sampling process of diffusion models can be computationally intensive. Achieving high-quality samples often necessitates a substantial investment of time and computational resources. This is because DDPM models the reverse process as Gaussian, which is valid only when the forward process adds a small amount of noise at each timestep. This means that the length of Markov chain should be large which slows down the sampling process. As a result, generating high-fidelity samples can require considerable amount of computations. It is crucial for researchers and practitioners utilizing diffusion models to be aware of these computational considerations. Subject to the resources and time available, one may need to make trade-offs between sample quality and computational efficiency. As mentioned above, DDPM suffers from slow sampling due to their reliance on the Gaussian assumption for the reverse Markov transition. This assumption necessitates the addition of small amounts of noise at each time step, leading to a long Markov chain and subsequent slow sampling. By removing the Gaussian assumption, we can overcome this limitation and introduce a denoising distribution that is no longer constrained to be Gaussian. This is where normalizing flows come into play. Normalizing flows have demonstrated an inherent capability to learn complex distributions effectively. By leveraging the flexibility and power of normalizing flows, we can effectively model the reverse diffusion process without being confined to Gaussian assumptions. This enables us to improve the sampling speed while retaining the desirable properties of diffusion models.

Normalizing Flows [10] represent a class of generative models renowned for their ability to learn the underlying probability distribution of a given dataset. Given a dataset, we want to learn the underlying distribution from which this data is generated. Normalizing flows offer a strategy for achieving this by transforming a simple and well-understood distribution, like a standard Gaussian distribution, into the desired target distribution. The core idea behind normalizing flows revolves around the use of a series of bijective (invertible) transformations. Each transformation maps one distribution to another, gradually transitioning the initial simple distribution into a more complex one that aligns with the target data distribution. These transformations are invertible, meaning that you can easily compute both the forward and backward transformations. Through these invertible transformations, normalizing flows enable the model to generate new samples from the transformed distribution.

A notable advantage of normalizing flow models lies in their ability to accurately compute the likelihood of the generated samples. Unlike other generative models such as Generative Adversarial Networks (GANs) or Denoising Diffusion Probabilistic Models (DDPM), which often struggle with likelihood estimation, normalizing flows excel in this aspect. Additionally, normalizing flows exhibit remarkable flexibility in terms of the transformations they can employ. They can incorporate a wide range of transformations, encompassing both linear transformations that preserve distribution shape and non-linear transformations that capture intricate relationships between variables. This adaptability enables normalizing flows to capture complex patterns and dependencies within the data, resulting in the generation of diverse and realistic samples.

We also note the limitations of normalizing flows. When designing normalizing flows, two important restrictions must be considered. Firstly, every transformation within the normalizing flows framework must be invertible. This requirement ensures that the flow can be reversed, allowing for both sampling and evaluation of probability densities. Secondly, the Jacobian determinant of each transformation should be tractable. This determinant plays a crucial role in updating the probability densities during the flow. These constraints significantly impact the choice of architectures and functions that can be used within normalizing flows to achieve the desired transformations while preserving the tractability of the underlying probabilistic computations.

In this work, we aim to build a diffusion model in which the reverse diffusion process is learned by normalizing flows. The diffusion framework, offers a principled approach to probabilistic modeling by formulating the problem as a sequence of continuous denoising steps applied to a simple base distribution, gradually transforming it into the desired target distribution. Denoising diffusion models have proven to be effective in capturing complex data distributions by iteratively applying noise and denoising steps to the observed data. However, DDPM requires adding small amount of Gaussian noise at each time step. This implies that the model requires a large number of time steps to effectively diffuse complex data into complete Gaussian noise. As a result, the reverse process needs to go through many denoising steps, slowing down the sampling. If one uses fewer time steps in the diffusions process, the denoising distribution in the reverse process is not Gaussian anymore and becomes a multi-modal distribution. To handle this multi-modal distribution, we propose to use normalizing flows. This integration allows for improved model expressiveness and faster convergence rates. Moreover, the combination of denoising diffusion and normalizing flows holds promising implications for various real-world applications. It can empower researchers and practitioners to tackle challenging problem like slow sampling process in image generation, speech synthesis, and other domains. Furthermore, the proposed integration can contribute to advancing our understanding of deep generative models and their capabilities in modeling real-world data

In our research, we adopt an evaluation framework comprising a set of generative model metrics namely FID Score and negative log likelihood. These metrics serve as crucial benchmarks for accurately assessing the performance and quality of our model. Our main objective is twofold. Firstly, we aim to leverage the diffusion model to generate high-quality images, ensuring that our outputs exhibit remarkable visual fidelity. Secondly, we strive to harness the capabilities of the normalizing flow framework to expedite the sampling process. Consequently, an additional focal point of our analysis involves comparing the sampling time required by our model against that of the diffusion model. By synergistically combining the strengths inherent in the diffusion model and the normalizing flow framework, we endeavor to strike a harmonious balance between image quality and sampling efficiency. Our ultimate goal is to demonstrate that our model not only produces visually captivating results but also significantly reduces the time required for image generation.

To achieve this, we conduct an examination of the FID Score—an essential metric providing insights into the visual similarity between the images generated by our model and authentic images sourced from the target distribution. Finally, we utilize the negative log likelihood metric to ascertain our model's proficiency in accurately estimating the likelihood of generating a given image. By minimizing the negative log likelihood, we actively enhance the overall quality and cohesiveness of our generated samples, bolstering their resemblance to real-world images.

Chapter 2

Background

2.1 Preliminaries

2.1.1 Gaussian Distribution

A well-known probability distribution, commonly referred to as the Gaussian distribution, finds frequent application in modeling real-world phenomena. This distribution is often characterized by its symmetrical shape, resembling that of a bell curve.

In simple terms, the Gaussian distribution represents a collection of values where the majority of data points tend to cluster around a central average value. The spread of data points gradually diminishes as one moves away from the center, resulting in fewer points towards the tails of the distribution. To define this distribution, two key parameters are used: the mean (μ) and the standard deviation (σ). The mean signifies the central value towards which the data is inclined to cluster, while the standard deviation reflects the extent of dispersion.

Mathematically, the Gaussian distribution's probability density function (PDF) can be expressed as follows: where,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.1}$$

where, x denotes a specific value at which the PDF is evaluated. μ and σ are the mean and standard deviation of the distribution respectively. σ controls the spread or variability of the data.

The Gaussian distribution exhibits a symmetrical shape around its mean. The mean serves as the peak of the distribution, and the standard deviation governs the width. A smaller standard deviation results in a taller and narrower curve, while a larger standard deviation gives rise to a shorter and wider curve.

2.1.2 Change of Variables Formula

A mathematical method called "change of variables" [11] enables us to change one set of variables into another. It is a foundational idea that is applied in many disciplines, such as probability theory and statistics. The purpose of changing the variables in a problem is to reframe it in terms of new variables that can either make analysis easier or give a deeper understanding of the issue at hand. The alteration of variables is particularly helpful for modifying probability distributions in probability theory. When we change the variables, we swap out the old ones for new ones that are connected by a mapping function. This transformation may involve numerous variables and be nonlinear.

The probability density functions (PDFs) of the original variables and the transformed variables are related using the change of variables formula. Let's say we have two random variables, X and Z, where X represents the transformed variable obtained through the use of a mapping function, f(Z), and Z represents the original variable.

The following is the formula for changing the variables:

$$p_X(x) = p_Z(f^{-1}(x)) \cdot |J| \tag{2.2}$$

where, $p_X(x)$ and $p_Z(z)$ represent the PDFs of the transformed variable X and the original variable Z, respectively. $f^{-1}(x)$ denotes the inverse of the mapping function f. $|J|$ represents the determinant of the Jacobian matrix, denoted by the vertical bars.

The Jacobian matrix is decsribed as a matrix of partial derivatives. The Jacobian matrix is given by the following formula:

$$J(f) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{2.3}$$

where x is a vector of input variables, and f_i is the i-th component of the function f. The Jacobian matrix is a square matrix, with n rows and n columns. The entry in the $i-th$ row and $j-th$ column of the Jacobian matrix is the partial derivative of $f(x)$ with respect to x_j, evaluated at x.

2.1.3 KL Divergence

KL Divergence, short for Kullback-Leibler Divergence, is a measure of the difference between two probability distributions. It is often used in statistics and information theory to quantify how one distribution differs from another.

The KL Divergence between two probability distributions, let's call them P and Q, is denoted as $KL(P||Q)$. It is defined as the expected value of the logarithmic difference between the probabilities of the two distributions:

$$KL(P||Q) = \int P(x) \log \left(\frac{P(x)}{Q(x)} \right) dx \tag{2.4}$$

In this formula, P(x) and Q(x) represent the probability densities (or probability mass functions) of the two distributions for a given value of x. The integral sums up the contribution of each value of x to the overall divergence.

KL Divergence has several important properties. KL Divergence is always non-negative. It equals zero if and only if P and Q are identical, meaning they have the same probabilities for all values of x. KL Divergence is not symmetric, which means that $KL(P||Q)$ is not necessarily the same as $KL(Q||P)$. In other words, the divergence from P to Q may not be the same as the divergence from Q to P. Although it is called“divergence,” KL Divergence is not a distance metric. That is, $KL(P||R)$ is not always less than or equal to $KL(P||Q) + KL(Q||R)$.

2.2 Diffusion Models

Diffusion models are a class of deep generative models that have gained significant attention in the realm of machine learning, specifically within the domain of computer vision. These models are designed to generate new data samples, such as images, based on an understanding of patterns and structures in existing data. The key idea behind diffusion models is to model the process of gradually transforming an initial distribution into a simple target distribution. This process is often referred to as “diffusing” the data. By modeling this diffusion process, the model can generate new samples by reversing the diffusion process. One of the notable advantages of diffusion models is their ability to generate high-quality samples with fine details and diversity. They have been successful in generating realistic images, exhibiting impressive generative capabilities. Additionally, diffusion models have been applied to various tasks in computer vision, such as image generation, image super-resolution, image inpainting, and image editing.

A diffusion model is based on a two step process as shown in Figure 2.1: the forward diffusion process, where Gaussian noise is gradually added in small time steps with the aim of "diffusing" the data so that the data distribution matches the Gaussian distribution and the reverse process, where a neural network learns to gradually reverse the diffusion process and generate the original input. The forward process begins with an image from a data set which is slowly transformed into a noisy image by adding small amounts of noise at different time steps. The noise is usually chosen as Gaussian noise. As the forward process continues, the input image is progressively destroyed until it reaches a Gaussian distribution. Due to the process of adding noise in small progressive timesteps, multiple steps are involved in the forward diffusion process.

During the reverse process, the aim is to reverse the process taken place during forward process. The goal is to recreate the original image from the noisy image by reversing the steps taken in the forward process. This is done by progressively removing the noise step by step that was added in the forward process. By removing noise at each timestep, the original image is gradually reconstructed. This means that during the inference or generation phase, new images are generated by incrementally reconstructing them, starting from random white noise. To remove noise from the image, a neural network is employed. The neural network estimates the amount of noise that has to be removed at each timestep. Typically a U-Net architecture [38] is used as the denoising neural network. U-Net is a particular type of neural network architecture that is commonly used in image segmentation tasks. By using such a network, the diffusion model can successfully maintain the dimensions and significant aspects of the original image while eliminating the noise. Denoising Diffusion Probabilistic Models (DDPM) [21] is a type of probabilistic generative model based on the above-described diffusion framework.

Let $x_0 \sim q(x)$ be a data sample belonging to $q(x)$ and $\pi(x)$ be a simple distribution (e.g., Gaussian) for Diffusion models. DDPM builds the forward process that converts any complex data from q to a sample from π which is any simple distribution, in our case, Gaussian distribution using a Markov Chain [41]. x_0 denotes the original sample that is uncorrupted. The goal of DDPM is to model p_θ that learns the reversal of this diffusion process.

Forward process: The forward process of the diffusion model is a fixed Markov chain process where noise is added gradually to a data point x_0 at different timestep t. This noise is added according to a variance schedule β. The Markov chain forward process is also called approximate posterior $q(x_{1:T}|x_0)$ [42].

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.5}$$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{2.6}$$

Here T is the total number of diffusion time steps involved, I is the Identity matrix and $\mathcal{N}(x; \mu, \sigma)$ is standard normal distribution with mean μ and variance σ. [42] mentions a special property of the forward process:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha}_t}x_0, (1-\overline{\alpha}_t)I) \tag{2.7}$$

where $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \frac{1}{t}\sum_{s=1}^{t} \alpha_s$. This property creates a closed-form solution allowing us to sample x_t for any t:

$$x_t = \sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon, \tag{2.8}$$

where $\epsilon \sim \mathcal{N}(0, I)$. As mentioned above, noise is added according to β which is the variance schedule. The choice of β or the variance schedule is an important consideration in the forward process. β decides how the addition of noise level changes over time in diffusion models. The variance schedule is chosen such that the value of β_T, which is the value of β at the last timestep T, approaches 0. This ensures that equation 2.6 reaches pure Gaussian distribution with a mean of 0 and standard deviation of 1 at time step T. Additionally, if we keep the values of β very very small, the forward process will be similar to the reverse process. Intuitively, if we go from x_{t-1} to x_t by adding a very very small amount of Gaussian noise, it becomes more likely that x_{t-1} comes from a region close to where x_t is observed. This allows us to model this region with a Gaussian distribution, as the small step ensures that the nearby regions have similar statistical properties [41]. The β values in the forward process can be kept constant or can be learned via reparameterization. In DDPM implementation, β values are kept constant increasing linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.

Reverse process: The reverse process of DDPM is a latent variable model parameterized as $p_\theta(x_0)$. The reverse process is modeled as the joint distribution $p_\theta(x_{0:T})$ of $p_\theta(x_0)$. The transitions of the reverse process begin at $t = T$ and is $p(x_T) = \mathcal{N}(x_T; 0, I)$, which is pure Gaussian noise.

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{2.9}$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{2.10}$$

The reverse process $p_\theta(x_{t-1}|x_t)$ is modeled through a neural network that predicts the mean $\mu_\theta(x_t, t)$ and the covariance $\sigma_\theta(x_t, t)$ after receiving a noisy image x_t and time step t as the input.

Loss function: For the loss function of diffusion models, KL divergence is used to compare the probability distribution of the two probabilities mentioned above: $p_\theta(x_{t-1}|x_t)$ and $q(x_{t-1}|x_t, x_0)$. The reverse process model p_θ is the conditional distribution of the previous state given the current state, as modeled by the diffusion model, and is not tractable. The forward posterior q is a posterior distribution of the previous state given the current state and the initial state x_0 and is tractable [21]. In an ideal scenario, when training a neural network, we would want to maximize the likelihood of the model assigning a high probability, $p_\theta(x_0)$, to each training example, x_0. However, the challenge is that directly computing $p_\theta(x_0)$ is difficult because it requires considering all possible reverse trajectories. To overcome this challenge, a solution proposed in the literature is to minimize a variational lower bound of the negative log-likelihood instead. This approach provides an approximation that is easier to compute. This formulation provides a way to estimate the negative log-likelihood by introducing a new term called the "variational lower-bound". The goal becomes minimizing this lower bound rather than directly maximizing the likelihood. By minimizing the variational lower bound, we are effectively training the neural network to optimize an approximation of the negative log-likelihood. This approximation allows for more tractable computations during the training process [6].

The authors use KL Divergence to compare $p_\theta(x_{t-1}|x_t)$ and $q(x_{t-1}|x_t, x_0)$ and thus arriving at the following formula:

$$\left[\mathrm{D_{KL}}\left(q(x_T|x_0)||p(x_T)\right) + \sum_{t>1} \mathrm{D_{KL}}\left(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)\right) - \log p_\theta(x_0|x_1) \right] \tag{2.11}$$

where $q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}t(x_t, x_0), \tilde{\beta}tI)$,

with $\tilde{\mu}t(x_t, x_0) := \sqrt{\bar{\alpha}t - 1\beta_t}\frac{1-\bar{\alpha}tx_0}{1-\bar{\alpha}t-1} + \sqrt{\alpha_t(1-\bar{\alpha}t-1)}\frac{1-\bar{\alpha}tx_t}{1-\bar{\alpha}_{t-1}}$ and $\tilde{\beta}t := \frac{1-\bar{\alpha}t-1}{1-\bar{\alpha}_t}\beta_t$. In order to simplify the above loss function, the DDPM authors [21] have divided the loss function into three parts for ease of understanding. The KL Divergence between $(q(x_T|x_0)$ and $p(x_T))$ is termed as L_t. Since the beta values in DDPM are set to constant and linearly increasing from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$, the term L_t remains constant.

The KL Divergence between $(q(x_{t-1}|x_t, x_0)$ and $p_\theta(x_{t-1}|x_t))$ is termed as L_{t-1} and the last term which is the negative of log of reverse process probability of x_0 given x_1 is termed as L_0. For L_{t-1} term, DDPM assumes,

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)), for 1 < t \leq T \tag{2.12}$$

Where DDPM sets $\Sigma_\theta(x_t, t) = \sigma_t^2 I$.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I) \tag{2.13}$$

The σ^2 was set to β_t by the authors because the authors found that it was the best choice for $x_0 \sim \mathcal{N}(0, I)$. With the above assumption of $p_\theta(x_{t-1}|x_t)$, a new loss function is obtained:

$$L_{t-1} = \mathbb{E}\left[\frac{1}{2\sigma_t^2}|\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)|^2\right] + \text{const} \tag{2.14}$$

Which says that the model μ_θ predicts noise. The authors of DDPM [21] have mentioned that there is a potential of predicting $x0$ which is the initial image not corrupted by any noise but the authors found it to lower the quality of samples generated. In this paper, we are trying to predict the image $x0$ directly by using the normalizing flow model instead of the diffusion model for our reverse process.

Equation 2.14 can be further expanded by reparametrization and using closed form equation in place of xt. This leads to a simplified loss function given used in DDPM as given below:

$$\mathbb{E}_{x_0,\epsilon}\left[\left|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t\right)\right|^2\right] \tag{2.15}$$

The above loss function calculates the MSE between actual noise and predicted noise for any image x_t. The DDPM paper showed the simplified loss function to perform better.

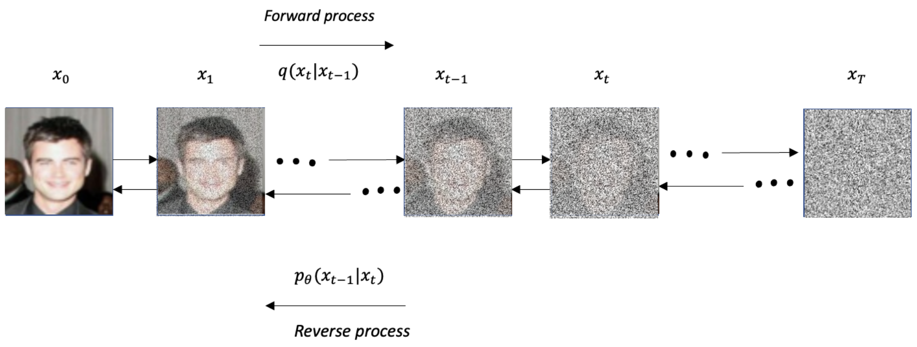

Figure 2.1: Forward and reverse process of the diffusion model

2.3 Normalizing Flows

A Normalizing Flow is a potent framework that harnesses a sequence of reversible and differentiable mappings to convert a basic probability distribution, like a Gaussian distribution, into a more intricate and adaptable distribution. This technique has gained considerable attention in recent years due to its effectiveness in modeling complex data distributions and accomplishing various probabilistic tasks.

The procedure of a Normalizing Flow involves employing a chain of transformations to modify a simple distribution and convert it into a complex distribution, or vice versa. Each transformation in the sequence is meticulously designed to be reversible, which means it possesses a corresponding inverse transformation that can revert the modified sample back to its original distribution.

Let Z be a random variable $Z \in \mathbb{R}^D$ having a tractable probability distribution p_Z and let X be an observed variable belonging to a dataset D. The normalizing flow uses a set of invertible functions such as f and the change of variable formula to calculate the density of x in terms of the latent variable z [10].

A series of invertible transformations $f_1, f_2, ..., f_K$ that map z to x, looks like this:

$$\begin{tikzcd} x \arrow[r, leftrightarrow, "f1"] & h_1 \arrow[r, leftrightarrow, "f2"] & h_2 \arrow[r, leftrightarrow, "f3"] & h_3 & \dots \arrow[r, leftrightarrow, "fK"] & z \end{tikzcd}$$

where, h_i represents latent variables for each transformation i. The complete set of transformations is given by the formula;

$$x = f_K \circ f_{K-1} \circ ... \circ f_1(z) \tag{2.16}$$

The change of variables formula for given random variables X and Z, that are related by a mapping function f such that

$$X = f(Z)$$

and

$$Z = f^{-1}(X)$$

gives the distribution of X as [10]:

$$p_X(x) = p_Z(f(x)) \left| \frac{\partial f(x)}{\partial x^T} \right| \tag{2.17}$$

Adding log on both sides of the above equation, we have [10]:

$$\log(p_X(x)) = \log(p_Z(f(x))) + \log\left|\frac{\partial f(x)}{\partial x^T}\right| \tag{2.18}$$

where $\frac{\partial f(x)}{\partial x^T}$ is the Jacobian of f at x.

Normalizing flows are trained by maximizing the total log-likelihood $n \log p_X(n)$ with respect to the parameters of the neural network that maps the function f.

In the context of Normalizing Flows, bijections, which are complex non-linear and invertible functions, need to be easy to compute, invert and to calculate the determinant of their Jacobian. There are different types of flows that can be constructed to implement the bijective properties of normalizing flows. The simplest of all is the Elementwise Flows which are transformations where each element is a vector and is individually modified using a scalar function that is bijective. The most common types of flows are Linear flows and they are used as the base for more sophisticated flow types. Linear Flows involve using linear mappings to express correlations between dimensions in a dataset. This approach complements the limitations of elementwise operations, which cannot capture such correlations. Linear mappings can be defined as

$$g(x) = Ax + b \tag{2.19}$$

where x represents a vector of input dimensions, A is a parameter matrix of size D x D, and b is a parameter vector of size D. If the matrix A is invertible, the function becomes invertible as well. As mentioned earlier, linear flows are more generally used as building blocks for more complex flows like affine coupling flows. These flows leverage the concept of linear mappings to create more flexible and expressive transformations as will be shown below. It is worth noting that the determinant of Jacobian for linear flows is denoted as $det(A)$ and calculating $det(A)$ can be computationally expensive as it is of the order $O(D^3)$. To address this issue, different methods are employed to restrict the form of the matrix A.

For example, matrix A can be constructed as a triangular matrix whose determinant is the product of its diagonal [25].

Two key ideas in the study of normalizing flows are coupling flows and autoregressive flows. Coupling flows and Autoregressive flows are one of the most widely used flow architectures. The paper "NICE: Non-linear Independent Components Estimation" [9] introduced a special type of normalizing flow called the coupling flow. The key idea behind NICE is to factorize the Jacobian determinant of the transformation into a product of diagonal matrices. By doing so, the computation of the determinant becomes efficient, allowing for tractable likelihood estimation. This factorization is achieved by splitting the input variables into two groups and applying different affine transformations to each group alternately. In a general coupling layer, we have an input x, which is divided into two parts: $I1$ and $I2$. Let $I1$ represent the first d dimensions of x, and $I2$ represent the remaining dimensions.

The output of the coupling layer is denoted as y, which consists of two parts: y_{I1} and y_{I2}. The y_{I1} part simply retains the values of x_{I1} unchanged. The transformation applied to x_{I2} is defined by a function called the coupling law, denoted as g. This function takes x_{I2} and a mapping function $m(x_{I1})$ as inputs and produces y_{I2}.

Mathematically, we can express the above coupling layer as follows [9]:

$$y_{I1} = x_{I1}$$
$$y_{I2} = g(x_{I2}, m(x_{I1}))$$

Here, g is an invertible map with respect to its first argument given the second argument, which ensures the invertibility of the coupling layer.

In the case of coupling flows, how to split or partition x remains an important question.

Several works mention different ways of splitting. “Density estimation using RealNVP” [10] introduces the RealNVP (Real-valued Non-Volume Preserving) algorithm as a method for flexible and efficient density estimation using normalizing flows. The RealNVP architecture employs a class of coupling layers, where each layer divides the input into two parts and transforms only one part while leaving the other unchanged. In the context of the RealNVP algorithm, coupling layers are designed as affine coupling layers. In an affine coupling layer, the transformation applied to the $I2$ part is an affine transformation, which consists of a scale function (s) and a translation function (t). These functions take the $I1$ part as input and produce parameters for the affine transformation. The output vector of the coupling layer is obtained by applying the affine transformation to the $I2$ part and combining it with the $I1$ part [12].

One important property of the coupling layers is that the Jacobian determinant of the transformation can be efficiently computed. The Jacobian matrix of the coupling layer has a triangular structure, and its determinant can be computed as the product of the diagonal elements, which depend on the scale function (s). This property allows for efficient training of the model using the determinant-based loss function.

To implement the coupling layers, masked convolutions can be used. Masked convolutions utilize binary masks to selectively update specific dimensions of the input vector while keeping others unchanged. The masks are designed based on the local correlation structure of images. Two types of masks are employed: spatial checkerboard patterns and channel-wise masking. The scale and translation functions (s and t) in the coupling layers are implemented as convolutional neural networks. “Glow: Generative Flow with Invertible 1x1 Convolutions” [24] builds on NICE and RealNVP. The main idea behind Glow is to utilize invertible 1x1 convolutions as building blocks for constructing a flow-based generative model. The architecture of the Glow model comprises multiple levels, each consisting of several steps. At each level, the input is divided into multiple channels, enabling the model to capture diverse aspects and dependencies of the data distribution. Each step within a level consists of three key components: an actnorm layer, an invertible 1x1 convolution, and a coupling transform. The actnorm layer is responsible for normalizing the input data to zero mean and unit variance along each channel. This normalization step helps stabilize the training process and ensures effective learning of the transformation parameters. The invertible 1x1 convolution plays a pivotal role in the Glow model. It performs a convolution operation on the input data while maintaining invertibility. This feature allows the model to capture intricate dependencies between the input and output spaces without loss of information. The coupling transform is also used as a crucial element in the Glow model. It splits the input data into two parts: a fixed part and a transformed part. The fixed part serves as a conditioning input, while the transformed part undergoes modification through an affine transformation. This approach enables the model to efficiently capture local variations and details in the data distribution. Glow utilizes a multi-scale architecture. This means that the model operates at multiple resolutions or levels, where each level comprises a set of steps. The multi-scale architecture empowers the model to effectively capture both global and local structures present in the data distribution.

Another type of flow is the autoregressive flow which is based on the concept of the autoregressive property. The probability distribution of each variable is represented in autoregressive models as a function of the preceding variables. This implies that each variable in the model only depends on the variables that came before it in a specific order. The ability to decompose a complex probability distribution into a series of conditional distributions that are easier to understand can be modeled using neural networks thanks to the autoregressive property. In particular, autoregressive normalizing flows employ a series of invertible neural networks, each of which transforms a subsequent variable in dependence on earlier ones. Autoregressive flows, much like coupling flows, use the same functional form which is a bijective function that has an input from one part of the space, and the parameters of the bijective function are determined based on the other part of the space. This bijective function is called the coupling function in both coupling flows and autoregressive flows.

There are many types of coupling functions in use in many of the literature stated above. One of the most common types of coupling functions which has been stated above several times is the affine coupling function used in “NICE” and “RealNVP” and many other literature. Spline is another type of coupling function commonly used. A spline is a type of function that is defined by a series of points called knots. The spline is constructed by connecting these knots using polynomial or rational functions. To create a useful coupling function, the spline should be monotone, which means that it consistently increases or decreases as we move along the spline. This monotonicity is achieved by ensuring that the x-values of the knots increase and the corresponding y-values also increase. Typically, splines are defined within a specific interval, such as a compact interval, to limit their range of values [32].

In the context of rational quadratic spline flows, a specific type of spline called a rational quadratic spline is utilized. [12] introduced the concept of rational quadratic spline flows, which involve modeling the coupling function as a monotone rational quadratic spline on an interval while behaving as the identity function outside of this interval. The construction of the spline involves specifying K+1 knots and their derivatives at the inner points. These knot locations and derivatives are parameterized as the output of a neural network, following a commonly used method in spline construction. The rational quadratic spline functions used in this approach are advantageous because they are easily differentiable and analytically invertible. The spline is divided into K bins, with each bin being defined by a monotonic rational quadratic function. A rational quadratic function can be expressed as the division of two quadratic polynomials. The spline ensures that it passes through the specified knots and the derivatives at the knots match the desired values. The transformation outside the interval is kept as the identity function, resulting in linear “tails” that allow for unconstrained inputs. The monotonicity, differentiability, and invertibility of the rational quadratic spline make it suitable for constructing coupling functions in the context of normalizing flows.

[12] states that the rational quadratic spline flows (RQ-NSF) can be used as a replacement for affine or additive transformations in normalizing flows. When used alongside alternating invertible linear transformations, they constitute a category of normalizing flows referred to as rational-quadratic neural spline flows (RQ-NSF). These flows can incorporate both coupling layers (RQ-NSF(C)) and autoregressive layers (RQ-NSF(AR)), offering improved flexibility compared to other transformation methods [12].

Chapter 3

Methodology

3.1 Motivation

While diffusion models have made notable progress in improving sample quality, it is important to acknowledge that these models can be computationally demanding during the sampling process. Generating high-quality samples often necessitates a substantial investment of time and computational resources. This is due, in part, to the complexity of the U-Net architecture employed by DDPM, which is known for its large number of parameters. The computational demands of diffusion models can be attributed to several factors. First, the sampling process involves iteratively denoising an initial noisy sample sampled from a known distribution like Gaussian. This denoising process requires multiple steps and involves complex computations, which can significantly increase the time and computational resources required. The computational demands of diffusion models arise from the nature of the sampling process and the necessity for multiple transformation steps. In diffusion models, the goal of the forward process is to transform an initial unknown data distribution into a known distribution, typically a Gaussian distribution. This transformation is achieved by iteratively adding small amounts of Gaussian noise to the data distribution over multiple timesteps. The reason for using multiple timesteps instead of adding all the noise in a single timestep is twofold. Firstly, if all the noise were added at once, the reverse process of generating a sample by predicting all the noise in a single timestep would be highly challenging. By spreading the noise addition over multiple timesteps, the model can more easily capture and learn the complex dependencies between the data and the noise. Each timestep allows the model to gradually refine its understanding of the transformation process, making it more feasible to generate high-quality samples. Secondly, the spacing between timesteps is crucial for achieving the desired transformation. The reverse process of diffusion models assumes a Gaussian distribution. The time interval between timesteps needs to be small because removing a small amount of Gaussian noise at each timestep progressively leads to the emergence of a Gaussian distribution. If the timesteps were too far apart, the noise removal process would be too abrupt, and the resulting distribution might deviate significantly from the desired Gaussian distribution. As a result, diffusion models require a large number of timesteps, typically ranging to thousands, to effectively transform the data distribution. This leads to a longer Markov chain requiring more time for denoising it. Additionally, the U-Net architecture used in DDPM contributes to the computational burden. The U-Net is a deep neural network with a large number of layers and parameters. While this allows the model to capture intricate patterns and dependencies in the data, it also increases the computational complexity of both training and sampling. The extensive parameterization of the U-Net necessitates numerous computations during both forward and backward passes, which can be time-consuming.

To address these challenges and improve the efficiency of diffusion models, we propose leveraging a conditional normalizing flow with rational quadratic transformation introduced by the Neural Spline Flow (NSF) [12] to model the reverse diffusion process. This allows us to eliminate the use of U-Net architecture and also removes the dependency of using multiple small timesteps for denoising the noisy samples. By employing a normalizing flow model in place of U-Net we can learn the direct mapping from noisy sample to original data by using invertible transformations which then allows us to skip timesteps while denoising. The normalizing flow approach offers a promising alternative to the U-Net architecture, potentially reducing the computational complexity while maintaining the model's expressive power. For the implementation of our method, we use the rational quadratic transformation as defined by the Neural Spline Flow [12]. By incorporating the rational quadratic transformation into the diffusion framework, we aim to enhance the efficiency of sampling and mitigate the computational demands associated with diffusion models.

Thus in summary, in this thesis, we focus on implementing and evaluating our proposed methodology, which adds the strengths of the normalizing flow with the rational quadratic transformation from [12] to the diffusion model framework. By integrating normalizing flow in the reverse process of diffusion models, we aim to improve the computational efficiency of diffusion models without compromising their modeling capabilities. The rational quadratic transformation provides an alternative architecture for the transformation process, potentially reducing the computational complexity and accelerating both training and sampling procedures.

3.2 Algorithm

This section presents the algorithm proposed in this paper, which uses a conditional normalizing flow to model the reverse process of diffusion models. The goal is to create a new generative model that leverages the normalizing flow model to reduce sampling time in diffusion models. Our approach involves incorporating a conditional normalizing flow into the diffusion model framework. The motivation behind this is to accelerate the sampling process of the diffusion model by selectively skipping timesteps. As mentioned in Section 3.1, the

Algorithm 1: Training

```
repeat
    $x_0 \sim q(x_0)$;
    $t \leftarrow \text{Uniform}(\{1, 2, ..., T\})$;
    $\epsilon \sim \mathcal{N}(0, I)$;
    $x_t \leftarrow \sqrt{\alpha_t} \cdot x_0 + \sqrt{1 - \alpha_t} \cdot \epsilon$;
    $t_{\text{emb}} \leftarrow$
      sinusoidal_embedding($t$);
    $C_t \leftarrow x_t + t_{\text{emb}}$;
    Take gradient descent step on:
      $nll = f_\theta(x_0, C_t)$, where
      $f_\theta = f_1 \cdot f_2 \cdot \ldots \cdot f_n$;
until converged;
```

Algorithm 2: Sampling

```
$\hat{x_0} \leftarrow$ None;
$first\_iter \leftarrow$ True;
for $t = T$ to 1 step $k$ do
    where k = number of
      timesteps to skip;
      $\epsilon \sim \mathcal{N}(0, I)$;
    if $first\_iter$ then
        $x_t \leftarrow \epsilon$;
        $first\_iter \leftarrow$ False;
    else
        $x_t \leftarrow \sqrt{\alpha_t} \cdot \hat{x_0} + \sqrt{1 - \alpha_t} \cdot \epsilon$;
    $t_{\text{emb}} \leftarrow$
      sinusoidal_embedding($t$);
    $C_t \leftarrow x_t + t_{\text{emb}}$;
    $\hat{x_0} \leftarrow f_\theta^{-1}(\epsilon, C_t)$;
Output: $\hat{x_0}$
```

Figure 3.1: Training and Sampling Algorithms

stepwise addition of noise is crucial for the effectiveness of the diffusion model, as it enables the model to learn the underlying data distribution more effectively. The reverse diffusion process assumes a Gaussian distribution. To achieve the desired distribution, small intervals between noise addition and removal are required. Large time intervals or excessive noise addition can lead to a deviation from the desired distribution. Diffusion models thus require a large number of timesteps, leading to a longer Markov chain and more time for denoising and sampling.

To address these challenges, we propose incorporating a conditional normalizing flow into the diffusion model to model the reverse process of diffusion models. The normalizing flow we use in our implementation is trained on a dataset D conditioned on noisy samples x_t at a given timestep t. Unlike the existing approaches that use U-Net to learn the reverse transition dynamics, we incorporate a normalizing flow to enhance the modeling capabilities and achieve more effective generative results. Traditionally, a U-Net architecture in DDPM takes an input image and a corresponding noisy image x_t, along with its associated timestep t, to predict the noise added in the forward process. However, we depart from this approach and opt for a normalizing flow as the primary tool to learn the reverse transition process.

In our implementation, we start by feeding the dataset sample as the input to the normalizing flow. Let us denote a sample by $x_0 \in D$. Using the closed-form equation of the diffusion model we generate the noisy sample x_t for a given timestep t:

$$x_t = \sqrt{\overline{\alpha}_t} x_0 + \sqrt{1 - \overline{\alpha}_t} \epsilon, \tag{3.1}$$

where $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \frac{1}{t} \sum_{s=1}^{t} \alpha_s$, β is variance schedule according to which noise is added to the image at timestep t, x_0 is the original sample from dataset D and ϵ is noise added. We provide the normalizing flow with both x_t and t in the form of context information C_t, which is described in detail in section 3.2.1, which corresponds to the noisy image at timestep t obtained by the closed-form solution in the DDPM framework along with the original samples.

$$z = f_\theta(x_0, C_t) \tag{3.2}$$

where, $z \in \mathcal{N}(0, I)$ and f_θ is a composition of transformation functions given by:

$$f_\theta = f_1 \circ \ldots \circ f_{N1} \circ f_N \tag{3.3}$$

By incorporating this conditional setup, we enable the model to capture the intricate relationship between the original samples and the noisy image at the given timestep. The model is optimized over negative log-likelihood of the normalizing flow which completes the training process. The negative log-likelihood of the normalizing flow model is given by:

$$\log p_\theta(\mathbf{x_0}) = \log p_{\mathbf{Z}}(\mathbf{z}) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right| \tag{3.4}$$

where θ denotes the parameters of normalizing flow and $p_{\mathbf{Z}}$ is known distribution such as Gaussian and z is a sample from Gaussian.

The sampling process designed for our implementation is represented by Figure 3.2. It is designed in such a way that it fastens the sampling process of the diffusion model. This is done by ensuring that the underlying normalizing flow model has the ability to learn $p_\theta(\mathbf{x_{t-k}}|\mathbf{x_t})$, which allows to jump multiple timesteps backward. During the sampling process, we follow an iterative process over a reverse sequence of timesteps. The denoising process of our implementation consists of two steps. Firstly, we predict $\hat{x_0}$, which is the predicted sample at timestep t from x_t using Conditional Normalizing Flow (CNF). Secondly, we obtain $x_t - k$ from the predicted $\hat{x_0}$. The sampling process of our implementation takes in a fresh sample from the Gaussian distribution and context x_t and t and outputs a newly generated sample $\hat{x_0}$. We apply the closed-form solution from the forward diffusion process and reach a sample at timestep $t - k$ where k is the number of timesteps skipped. This process is repeated iteratively for a sequence of timesteps. This eventually leads to the generation of a new sample in a small number of timesteps as compared to the traditional diffusion model. It is important to note that x_t at the first timestep is fresh noise.

A difference between our sampling process and the sampling process employed by DDPM is that DDPM predicts noise whereas our approach directly predicts the denoised sample.

3.2.1 Contextual information

This section presents the process of generating contextual information for our proposed method, which is essential for effectively conditioning the normalizing flow model on the forward process of the diffusion model. The contextual information is derived using the

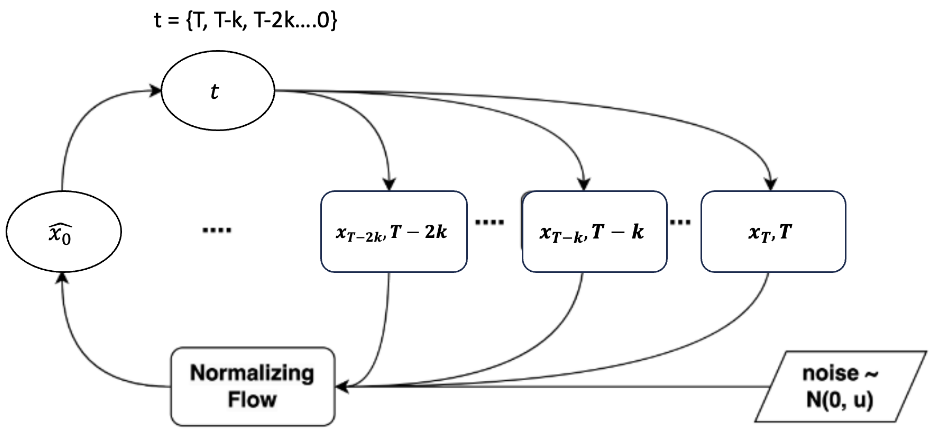

Figure 3.2: Sampling Process of our Implementation

closed-form equation of diffusion models, enabling us to incorporate the underlying dynamics into the data generation process.

Given a specific time-step t, the noisy data x_t is generated by simulating the forward process of diffusion models. This is achieved by adding noise sampled from a Gaussian distribution to the data. Mathematically, we can express this as the closed-form equation of the diffusion models as described in the previous section:

$$x_t = \sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon, \tag{3.5}$$

During the training process, the time-step t is randomly selected for the input data. Additionally, we transform and convert the time-step t into a sinusoidal embedding, providing the network with explicit time information. This embedding allows the model to effectively utilize and learn the temporal patterns present in the data. Mathematically, the sinusoidal embedding can be represented as:

$$t_{\text{embedding}} = \sin\left(\frac{2\pi t}{T}\right), \cos\left(\frac{2\pi t}{T}\right) \tag{3.6}$$

where T represents the maximum value of the timestep.

To incorporate the contextual information x_t and the transformed time-step $t_{\text{embedding}}$ into the model, we perform an element-wise addition, yielding a representation denoted as C_t. This merged representation serves as the input context for the normalizing flow model. Mathematically, this can be expressed as:

$$C_t = x_t + t_{\text{embedding}} \tag{3.7}$$

Furthermore, we have explored alternative techniques as shown in Figure 3.3, including convolutional neural networks and multi-layer perceptron, to merge the contextual information. These approaches provide flexibility in capturing complex relationships and dependencies between the input data and the noisy data at a specific timestep.

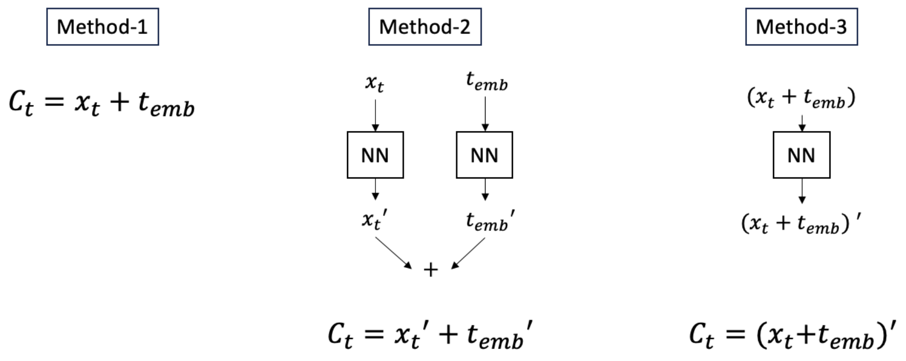

Figure 3.3: Different ways of adding context information

The processing of context information in conditional normalizing flow models can take place in various ways. In our proposed implementation, we utilize the glow model, which comprises an actnorm layer, an invertible 1×1 convolutional layer, and a rational quadratic spline coupling layer instead of an affine coupling layer. Within our framework, the conditional normalizing flow model takes two inputs as shown in Figure 3.4, real samples and context information which are noisy images at specific timesteps, denoted by x_t. To preprocess the real samples, we apply the squeeze transformation, followed by the actnorm layer and the 1×1 convolutional layer. Similarly, the context information undergoes the squeeze transformation which ensures that the context and input data are of the same shape. Subsequently, both the real samples and context information pass through a convolutional layer before entering the rational quadratic coupling layer. By integrating the contextual information, our model gains a comprehensive understanding of the data, leveraging both the observed inputs which are real samples, and the generated contextual information. This enhanced capability facilitates the exploration and exploitation of underlying patterns and dependencies, aiming to improve performance and accuracy in modeling the data.

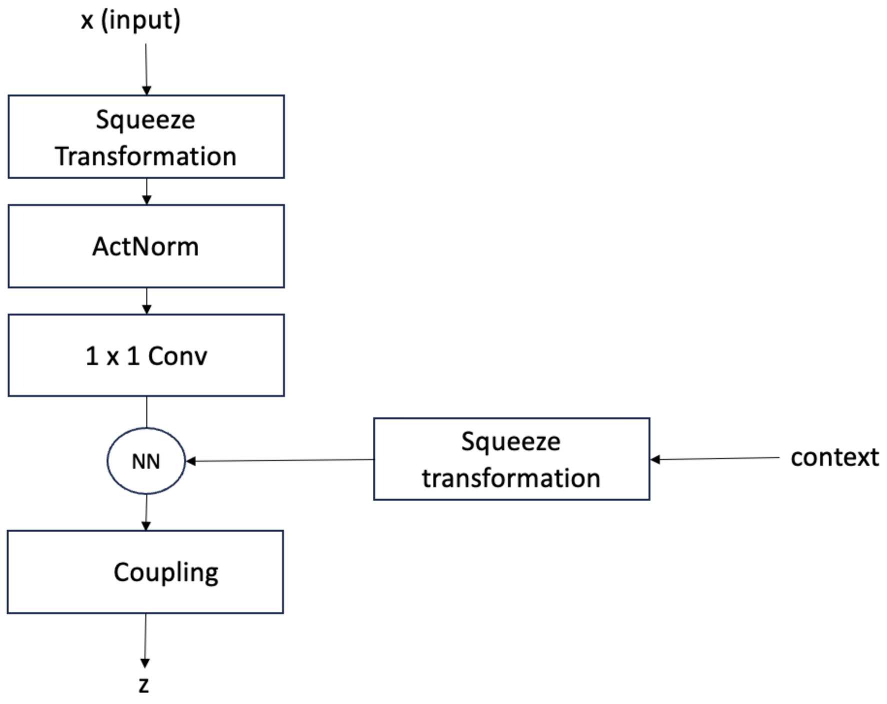

Figure 3.4: Conditional Normalizing Flows

3.3 Implementation Details

To implement our suggested methodology, we leveraged the code from two distinct repositories, namely the Denoising Diffusion Probabilistic Models (DDPM) repository [45] and the Neural Spline Flow repository [13]. By combining and adapting the code from these repositories, we were able to take advantage of the strengths offered by both models and tailor the implementation to suit the specific requirements of our research. In essence, we utilized the DDPM code to handle the forward process of the diffusion model in mainly the generation of noisy images using the closed-form solution, while the Neural Spline Flow code was employed to model the probability distribution function. The resulting implementation served as the foundation for conducting our experimental analysis. The Neural Spline Flow code incorporates the use of rational quadratic spline transforms, as outlined in the work [12]. Specifically, we adopted the architecture employed in the Glow model [24]. The transformation process aligns with the transformations used to create neural spline flow, and we opted for a multi-scale architecture. The number of flows within the model is determined by two key variables: "levels" and "steps per level". Each level comprises a squeeze transform, actnorm layer, an invertible 1 x 1 convolutional layer, and a number of transform steps corresponding to the specified steps per level. As previously mentioned, we utilized the piecewise rational quadratic coupling transform with linear tails for each step. We set a tail bound of 5 and generated a mid-binary split mask based on the number of channels in the dataset. Additionally, we incorporated the preprocessing transform provided by the neural spline flow repository, specifically selecting the real nvp preprocessing transform. This preprocessing transform maps the input data to the range [0, 1] before passing it through subsequent transformations and computations. This step ensures that the input data is appropriately scaled for optimal processing by the remaining transforms in the pipeline.

Chapter 4

Related Works

4.1 Diffusion Models

There are many works in the field of diffusion models that aim at solving the high sampling times of DDPM. Denoising Diffusion Implicit Models [42] (DDIM) are an extension of DDPM models where the forward process used in DDIM is Non-Markovian as opposed to the Markovian process in DDPMs. This ensures that noisy image sample from given timesteps in DDIMs can depend on their previous image, as in DDPM, and also on the very first timestep. This work is an effective implementation in the domain of reducing the sampling time of the diffusion models by reducing the number of timesteps. Our work is an extension of this mission of working on reducing the number of timesteps for diffusion models.

[33] propose further improvements in DDPM. They found that the performance of diffusion models in terms of log-likelihood can be significantly improved by learning the variance parameter. Allowing the model to learn the variance enhances its efficiency in generating samples. Instead of the simple mean squared error (MSE) loss function that DDPM uses, they use a hybrid loss function composed of the MSE loss function as defined by DDPM and the variational lower bound. The variational lower bound is controlled by a regularization

term. The authors introduce a new noise schedule that adapts to the local complexity of the data. By adjusting the noise level during the diffusion process, the model can effectively capture both low and high-frequency details in the generated images.

[28] introduced Bilateral Denoising Diffusion Models (BDDMS) to solve the problem of using a large number of timesteps. The authors of BDDMS acknowledge the formation of longer Markov chains in the forward process leading to a longer sampling time. Previous research such as [4, 26] have shown that changing the noise schedule in the sampling process and keeping it different from the noise schedule used in forward process can reduce the sampling time. These approaches followed a non-linear noise schedule where [26] used a fixed noise schedule and [4] used a grid search method to find the best one. BDDMS aims to directly learn the noise schedule for sampling. They also use a bilateral mode of training where forward and reverse process train together. They first use the forward process to sample from a distribution and obtain an initial sample. Then, they initiate the reverse process starting from that initial sample. Two networks, a score network, and a scheduling network are used to parameterize the forward and reverse processes. They were able to show a significant reduction in the sampling time but the use of two networks can add to the computational overload. As opposed to their method, our method uses only one network for modeling the reverse process. They also use a score network whereas we use a conditional normalizing flow.

Denoising Diffusion Restoration Models (DDRM) [23] is an approach for solving image restoration problems like denoising and improving image quality. DDRM generates good-quality results in a relatively shorter amount of time by employing a pre-trained diffusion model. The key idea behind DDRM is the development of an unsupervised method which does not require task specific training for any of the image restoration task. By using pre-trained diffusion model, DDRM produces diverse samples for different tasks relatively faster.

[39] also discuss the problem of requiring high computational resources and time while

training and evaluating diffusion models. To tackle this, they introduce Pyramid Diffusion Model. Their approach uses a diffusion model conditioned on positional embedding and generates higher-quality samples starting from coarser-quality samples. In their forward process, they provide real samples as well as the position of each pixel as conditional information. In their reverse process, instead of using a single full diffusion model for reverse sampling, they use a multi-scale approach. They employ a pyramid structure where the reverse sampling process starts from the lowest resolution and progressively moves to higher resolutions. This enables faster and more efficient sampling. In this approach, they solve the problem of slow sampling of diffusion models by using a conditional diffusion model whereas we use a conditional normalizing flow model. They condition their model on positional embedding whereas our model is conditioned on noisy samples at given timesteps.

4.2 Normalizing Flows

In recent years, there has been a growing interest in incorporating conditional information into normalizing flow models to enhance their capabilities in various domains. Several notable works have explored different approaches to leverage conditional normalizing flows. The work by the authors in [1] introduces StyleFlow, a significant contribution in the field of attribute-controlled image generation using conditional continuous normalizing flows. Previous methods, including unconditional GANs like StyleGAN, have limitations in controlling attribute-specific generation while maintaining image quality. However, StyleFlow overcomes these limitations by conditioning the generation process on attribute features. This allows for precise control and fine-grained editing of specific attributes, such as pose, expression, and age, without undesired effects on other attributes. Another notable work, CFLOW-AD [17], presents a real-time model based on a conditional normalizing flow framework specifically designed for anomaly detection with localization. The architecture of CFLOW-AD has a discriminatively pre-trained encoder along with a multi-scale generative decoder. The encoder extracts relevant features from the input data, while the generative decoders estimate the likelihood of encoded features at multiple scales, enabling anomaly detection and localization. The conditional normalizing flow model in our approach uses noisy samples of the original image as conditional information as opposed to various attributes of the image like most of the implementations follow.

[43] proposes a fully conditional Glow-based architecture for generating realistic images of street scenes given a semantic segmentation map. The authors suggest that this approach can be used to augment existing datasets with synthetic images, which can be beneficial for training visual semantic segmentation or object recognition systems. The paper discusses the challenges of collecting labeled visual data for training autonomous agents, such as driverless cars, and highlights the benefits of using generative models for data augmentation. It mentions that GANs and auto-regressive architectures have been widely used for image generation but have certain limitations. Normalizing flows, specifically, Glow, are introduced as a promising alternative that can generate high-quality images and have advantages such as stable training, explicit representation learning, efficient synthesis, and exact likelihood evaluation. The proposed architecture, Full-Glow, combines innovations from previous Glow-based models for conditional image generation. It consists of two parallel stacks of Glow, where all sub-steps are made conditional by inserting conditioning networks. The conditioning networks allow for the integration of source-domain information at the relevant level of hierarchy. The architecture includes conditional actnorm and 1x1 convolutions, which are also connected to the source Glow. The conditioning networks are built using convolutional and fully connected layers.

4.3 Hybrid of Diffusion Models and Normalizing Flows

The paper Diffusion Normalizing Flow [46] proposes a new methodology that combines diffusion models and normalizing flows. Diffusion Normalizing Flow (DNFs) use a diffusion process to convert a complex data distribution to a base distribution such as Gaussian and then use a normalizing flow to learn transformations from a base distribution to a complex dataset. DNFs do the two processes i.e. forward diffusion process and the normalizing flow process simultaneously. DNFs also use two neural networks for the two simultaneous processes. This could result in better modeling of complex target distributions and improved performance. The difference between our approach and the approach used in DNFs is that we do not use a neural network for the forward process making our process less parameter-heavy. We also use a conditional normalizing flow as opposed to the traditional normalizing flow making it easier for the neural network to learn features effectively.

"Generalized Normalizing Flows via Markov Chains" [18] presents a new framework for generative models that combines deterministic and stochastic normalizing flows. The framework is based on the idea of Markov chains. The authors argue that the Markov chain framework provides a number of advantages over traditional approaches to generative modeling, including increased flexibility, expressiveness, and mathematical rigor. The authors define a stochastic normalizing flow as a Markov chain where the transition kernel is a deterministic invertible function. This means that the Markov chain can be represented as a sequence of deterministic invertible functions, each of which transforms the state of the chain to a new state. This definition allows for more flexible and expressive modeling of complex probability distributions. This is because the Markov chain can be used to model the evolution of a random variable through a sequence of different probability distributions. A number of theoretical results that support the use of the Markov chain framework for generative modeling are presented. The authors also demonstrate the effectiveness of the framework on a number of real-world datasets.

[40] utilizes a combination of normalizing flow models and diffusion models to help flow-based models solve the problem of latent space mapping. Normalizing flows have limitations in fully mapping to a Gaussian distribution, as demonstrated in research [42]. While these models strive to approximate the target distribution, they may not achieve a perfect match. This limitation affects the model's ability to generate samples that cover the entire input space faithfully. [40] aims to use diffusion model along with normalizing flow models to effectively map the latent space to Gaussian distribution. This ensures that no approximations are generated while sampling from the transformed distribution leading to a better quality of generated samples.

Chapter 5

Experiments

To demonstrate the effectiveness of DiffusionCNF, we evaluated the model on 3 popular benchmark datasets: MNIST, CIFAR-10, and CelebA. MNIST dataset [29] is a collection of handwritten images. Each image in the dataset corresponds to a handwritten digit ranging from 0 to 9. It consists of 60000 training samples and 10000 test samples. Each image in the MNIST dataset is a gray scale image of size 28 x 28 pixels. CIFAR-10 dataset [27] is a collection of images belonging to 10 different classes. It has 60000 training samples where each class consists of 6000 samples. The CIFAR-10 dataset consists of RGB images with a resolution of 32 x 32 pixels. The CelebA dataset [31] consists of a collection of celebrity faces. There are 202,599 RGB samples in the dataset with each image being 178 x 218 pixels.

For our model, we used the Adam optimizer with a learning rate of 0.0001 during training. We trained our model for varying numbers of epochs depending on the dataset under consideration. For MNIST dataset, our normalizing flow model consists of 4 levels where each level is composed of 30 flow steps. Every flow step is a combination of actnorm, 1x1 convolution, and affine coupling transformation as explained in section 3.3. The number of features were set to 128. For the CIFAR-10 dataset, we employed 30 steps per level with 3 levels and 128 hidden features. The number of steps per level was set to 25 with 4 levels for the CelebA dataset and hidden features were set to 128. Table 5.1 summarizes the above training configuration.

Timesteps in the diffusion model control the number of steps over which noise is gradually added in the forward step in the diffusion model. In our implementation, we utilized a total of 100 timesteps for the sampling process, as opposed to the default 1000 timesteps used in traditional DDPM models. The selection of 100 timesteps allowed us to achieve efficient and effective sampling while reducing the computational burden. To generate the timestep sequence, we employed a uniform skip type, ensuring that the intervals between timesteps were evenly distributed. The sequence has 100 steps skipped in between.

Table 5.1: Training Configuration

Configuration	**Value**
Batch size (MNIST)	512
Batch size (CelebA)	64
Batch size (CIFAR-10)	150
Steps per level (MNIST)	30
Steps per level (CelebA)	25
Steps per level (CIFAR-10)	30
Level (MNIST)	4
Level (CelebA)	4
Level (CIFAR-10)	3
Learning Rate	0.0001
Transform type	Rational Quadratic Spline
Total diffusion timesteps for sampling	100
Optimizer	Adam

In terms of architecture, we followed the implementation details mentioned in the previous section. Additionally, we evaluated the performance of our model using two key metrics: Negative Log Likelihood (NLL) and Frechet Inception Distance (FID) [19]. Negative log likelihood measures the underlying distribution of the original data. It is a measure to determine how accurately the model predicts the original distribution of training data. A

Table 5.2: Experiment Results

Dataset	FID	nll(bpd)
MNIST	2.13	0.88
CELEBA	4.13	2.66
CIFAR-10	3.47	3.28

lower NLL indicates that the model can effectively model the data distribution and generate samples that closely match the real data distribution. We presented the nll in terms of bits per dimension (bpd). The FID Score quantifies the similarity between the distribution of generated samples and the real samples. It employs a pre-trained Inception network [44] to extract features and calculate the distance between the feature distributions. A lower FID score indicates better quality and diversity in the generated samples [19]. The FID score depends on the feature vectors that we use. In our case, all models have been evaluated by setting the feature vector length to 768. We used FID function provided by the torchmetrics library.

To further evaluate the performance of our model, a comparison was made with other models on the CelebA dataset and MNIST dataset. Table 5.3 and Table 5.4 display the results. Furthermore, we assessed the efficiency of our approach by measuring the sampling times for generating 25 samples for each experiment. These time measurements provided valuable insights into the computational efficiency and practical viability of our integrated approach. The analysis of all the experiments is discussed in the results section.

5.1 Results

Table 5.2 provides the performance metrics for our model on three different datasets: MNIST, CelebA, and CIFAR-10. For the MNIST dataset, our model achieved an FID score of 2.13 and an nll of 0.88 bits per dimension (bpd). An FID of 2.13 indicates a lower average

Table 5.3: Comparison with Different models on CELEBA

Models	FID	loss	sampling time
Our model	4.13	2.66 nll	34 secs
DDPM	2.94	0.03 mse	42 secs
RQ-NSF(C)	3.02	2.86 nll	0.22 secs

Table 5.4: Comparison with Different models on MNIST

Models	FID	loss	sampling time
Our model	2.13	0.88 nll	21.5 secs
DDPM	2.34	0.03 mse	29.9 secs
RQ-NSF(C)	3.48	0.89 nll	0.21 secs

distance between the generated samples and real images from the MNIST dataset. A nll value of 0.88 shows that the model was able to accurately generate the underlying data distribution of the original data. The quantitative results of the model demonstrate that our model generates good-quality results for the MNIST dataset. The lower FID score and lower nll value can also be attributed to the simple structure of the MNIST dataset consisting of only grayscale images. In terms of the visual quality, the samples generated by the model are easily identifiable as handwritten digits as shown in Figure 5.5. However, a few samples show minor irregularities or breaks in the digit shape. Further training may help alleviate the discontinuities observed in some samples. Overall, since this is a simple dataset our model was able to better capture it's underlying data distribution and this is reflected in the negative log-likelihood as well as the FID score of the generated samples. Further in this section, we will compare the outputs of our model on MNIST datset with other models providing analysis on the same.

On the CELEBA dataset, the FID score obtained was 4.13, and the nll was 2.66 bpd. These metrics suggest that the generated samples of the CelebA dataset are lower in quality compared to the MNIST dataset. The difference in the FID scores and the nll between the MNIST and the CelebA dataset can be associated with the relatively higher complexity of the CelebA dataset when compared to MNIST. The CelebA dataset consists of more complex and diverse images with varying backgrounds. These complexities may make the model to converge slowly resulting in a higher nll value than the MNIST dataset. Our model is able to generate samples that resemble faces, as seen in Figure 5.6. However, some distortion in the images are observed. As mentioned in Section 5, we utilized 25 steps per level in our experiments due to computational limitations. Due to the complexity of the CelebA dataset, more steps per level may increase the quality of generated samples and help in achieving a lower FID score and nll. Similarly, for the CIFAR-10 dataset, our model achieved an FID score of 3.47 and an nll of 3.28 bpd. The higher nll value suggests that the model faces challenges in accurately capturing the underlying data distribution present in the CIFAR-10 dataset. Visually, the images are hard to recognize, as shown in Figure 5.7, and seem to be more blurry compared to the other two datasets. It was observed that the model, when trained on the CIFAR-10 dataset, needed more number of epochs compared to CelebA or MNIST to reach convergence. This may be tackled by using a learning rate scheduler to facilitate smoother convergence. More efforts towards fine tunning the model specifically when trained on the CIFAR-10 dataset may prove to be valuable. The quality of samples generated by the model when trained on the CelebA dataset was better than the ones obtained by training on the CIFAR-10 dataset. The CelebA dataset consists of facial images with a high resolution pixels, providing a more detailed representation of facial features and textures. The higher resolution allows the model to capture fine-grained details, resulting in more realistic and visually appealing generated samples. On the other hand, the CIFAR-10 dataset comprises images with a resolution of 32 x 32 pixels, limiting the level of detail that can be captured by the model.

As mentioned in section 5, we performed experiments to compare the performance of our proposed model with that of DDPM [21] and RQ-NSF(C) [12]. Table 5.3 and Table

5.4 show FID, nll, and sampling times of different models. DDPM achieved a lower FID score of 2.94. This indicates that DDPM has a better ability to capture the underlying distribution of the CelebA dataset. One reason for DDPM's low FID score can be attributed to its repetitive denoising process. However, this leads to a trade-off between the quality of samples generated and the sampling time. DDPM generates 25 samples in 42 seconds which indicates it is the slowest among all models due to its repeated denoising process. On the other hand, RQ-NSF(C) exhibited the fastest sampling time among the compared models, generating samples in just 0.22 seconds. RQ-NSF(C) obtained an FID score of 3.02, placing it in between our model and DDPM in terms of image quality. When comparing these models with our proposed model, we observe that our model performs comparably to DDPM in terms of FID scores on the CelebA dataset. Normalizing flow models typically require large number of transformations and thus the higher FID score can be bought down by training on a larger number of flows. Our model has an advantage in terms of sampling time, as it generates 25 samples in 34 seconds compared to DDPM's 42 seconds. This suggests that our model achieves a better balance between generative performance and computational efficiency, making it a favorable choice when time constraints are important. Moreover, it is important to consider the trade-offs between these models. DDPM excels in terms of image quality and reconstruction performance but requires a longer sampling time, indicating a higher computational overhead. RQ-NSF(C) demonstrates impressive sampling speed but may sacrifice some accuracy in capturing the dataset's statistical properties. In contrast, our proposed model strikes a balance between FID scores and sampling time making it a promising choice for applications that require a satisfactory trade-off between efficiency and quality. It is worth noting that our model utilized a total of 100 sampling timesteps, whereas DDPM used 1000 timesteps. However, further experimentation holds the potential to achieve both improved sample quality and reduced sampling time. We can also investigate alternative approaches to incorporate contextual information into normalizing flows for improving the sample quality. Other methods such as refining the architecture or incorporating additional contextual cues, may lead to higher-quality samples and more efficient sampling procedures.

In addition to the quantitative evaluation, we have also provided the comparison of samples generated from the three different models in order to provide a visual representation of the comparison. Figure 5.1 and Figure 5.3 shows the generated samples from our model, i.e DiffusionCNF and from the DDPM model for CelebA dataset and MNIST dataset respectively. Figure 5.2 and Figure 5.4 shows the generated samples from our model, i.e DiffusionCNF and from the RQ-NSF(C) model for CelebA dataset and MNIST dataset respectively.

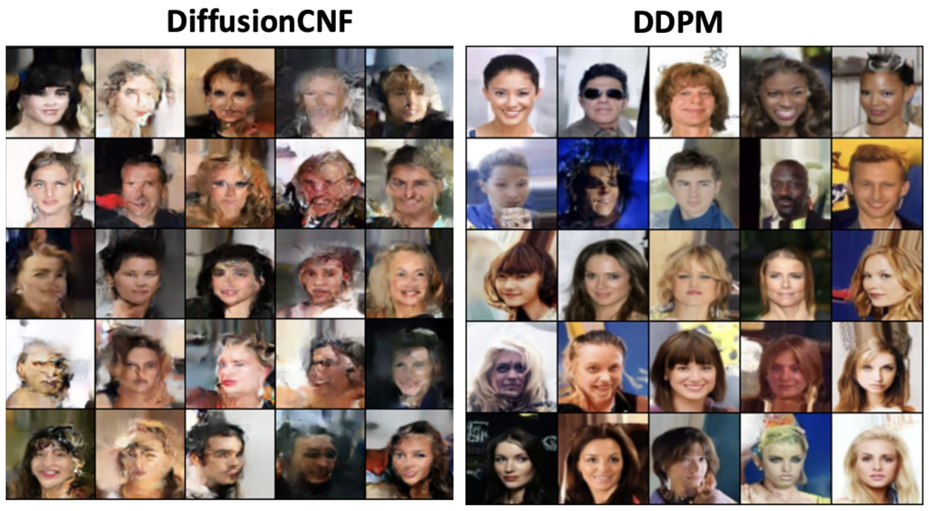

Figure 5.1: Comparison of generated samples from DiffusionCNF of CELEBA data with generated samples from DDPM

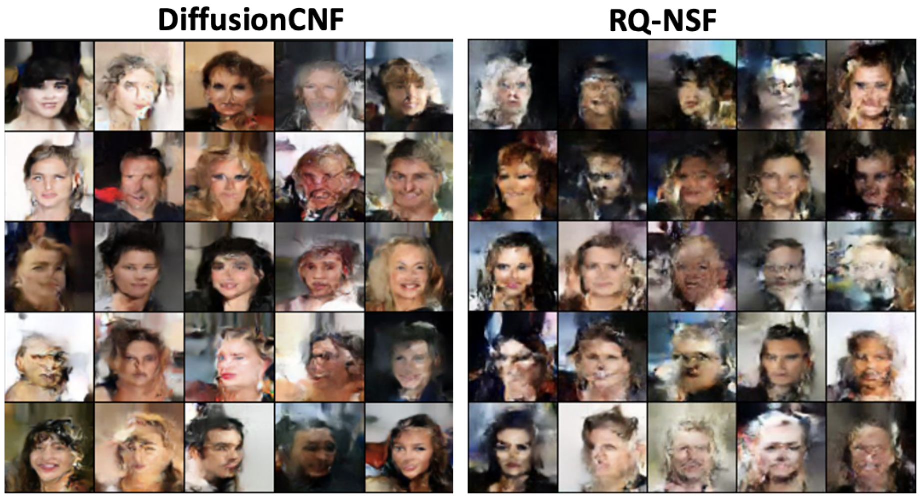

Figure 5.2: Comparison of generated samples from DiffusionCNF of CELEBA data with generated samples from RQ-NSF

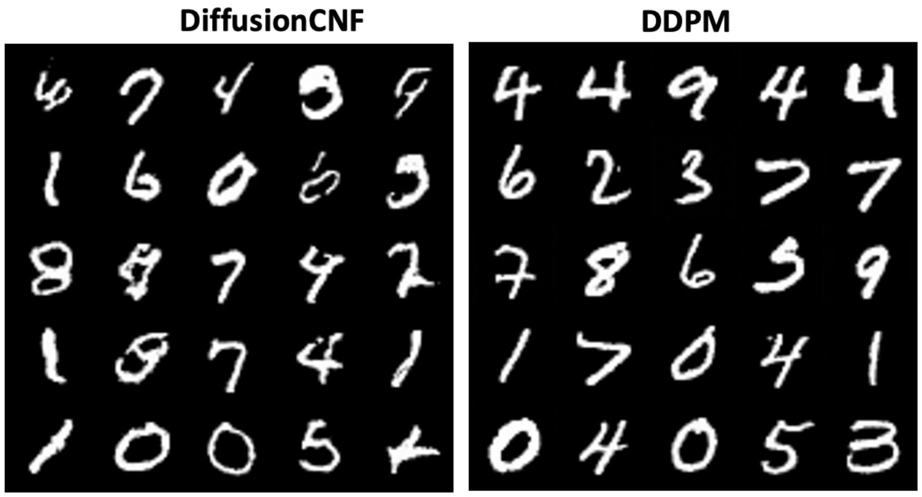

Figure 5.3: Comparison of generated samples from DiffusionCNF of MNIST data with generated samples from DDPM

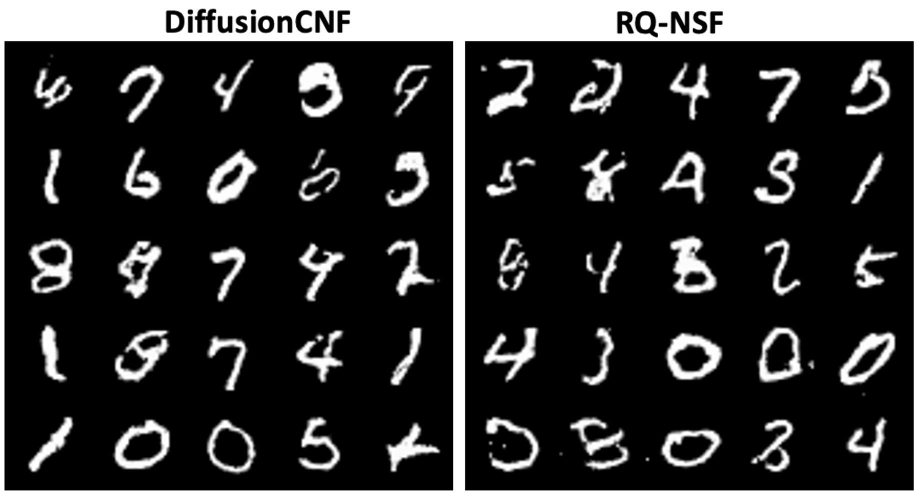

Figure 5.4: Comparison of generated samples from DiffusionCNF of MNIST data with generated samples from RQ-NSF

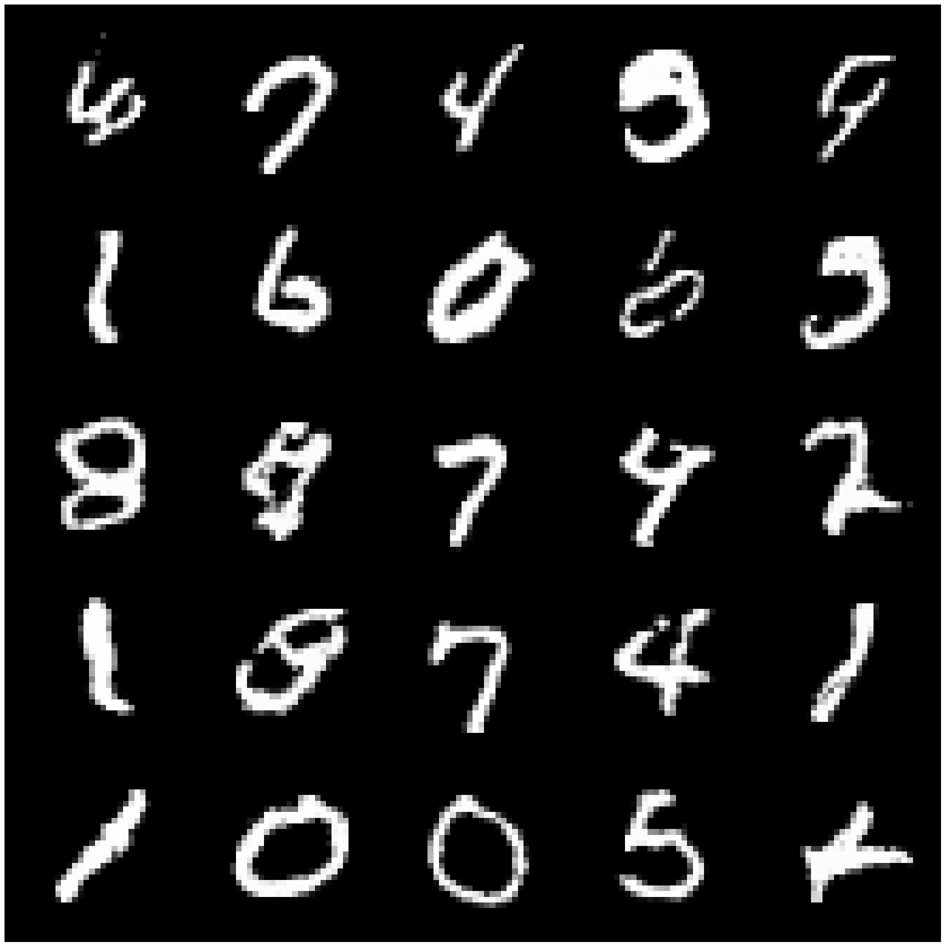

Figure 5.5: Generated Image Samples of MNIST data from our Implementation

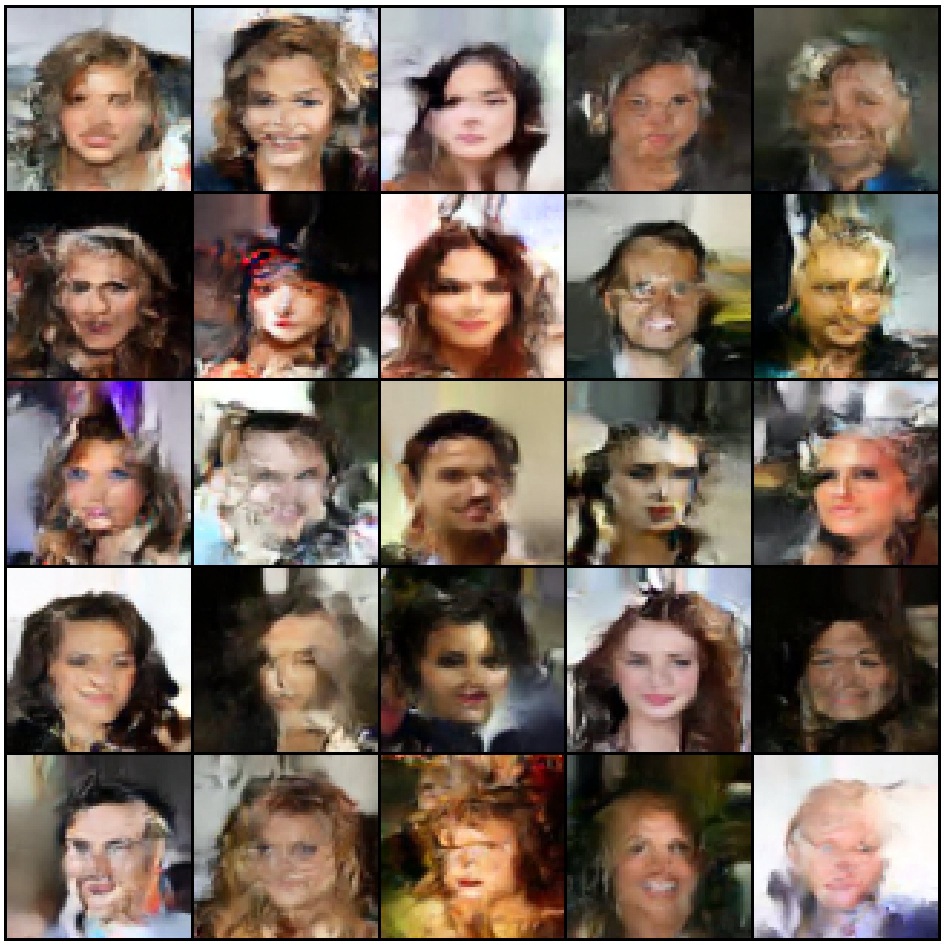

Figure 5.6: Generated Image Samples of CELEB-A data from our Implementation

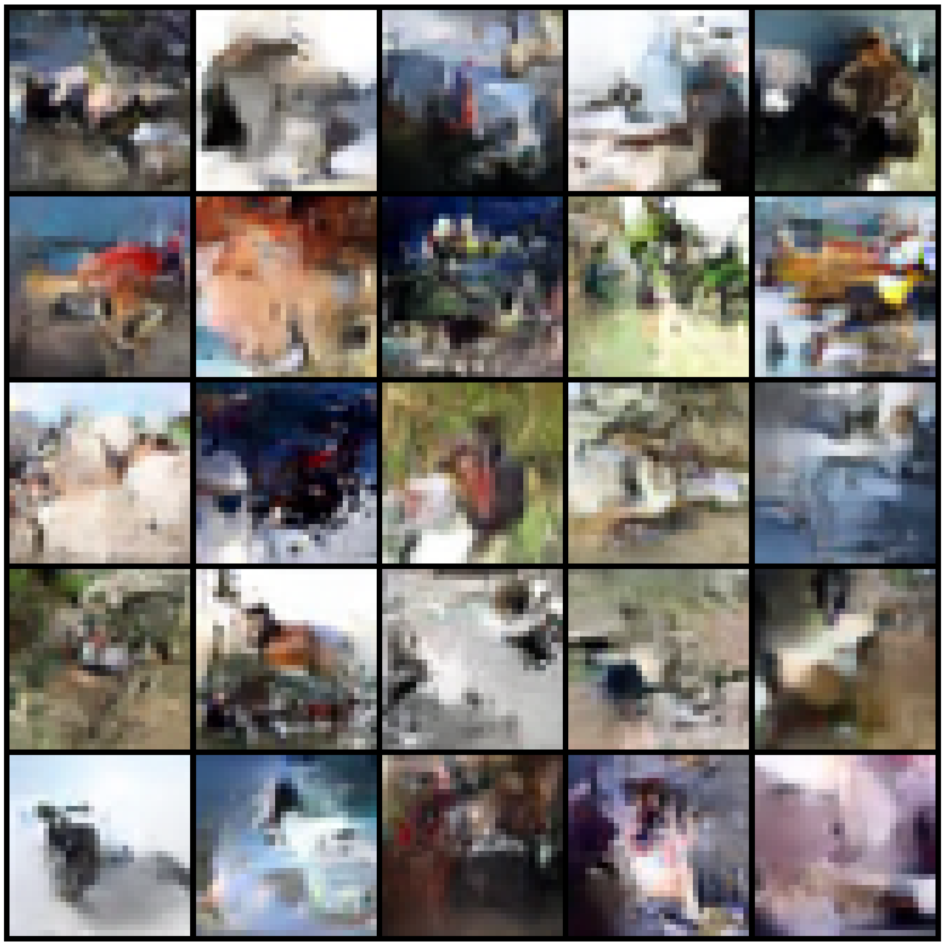

Figure 5.7: Generated Image Samples of CIFAR-10 data from our Implementation

Chapter 6

Discussion

In conclusion, our implementation aims to leverage the strengths of normalizing flows within diffusion models to address the limitation of slow sampling time. By integrating a normalizing flow into the reverse process of the diffusion model, our objective is to significantly reduce the time required for image generation.

Moving forward, there are several avenues for further experimentation and refinement of our new model. One potential direction is to explore better methods of incorporating contextual information by enhancing the formulation of C_t, which combines x_t and t. Investigating alternative approaches to incorporate C_t into the normalizing flows could potentially lead to improved modeling of the reverse diffusion process.

Furthermore, exploring different types of normalizing flows within the diffusion framework is worth considering. There are various types of normalizing flows, such as RealNVP, Glow, or MAF, each with its own advantages and characteristics. Evaluating the performance of different types of normalizing flows and their compatibility with the diffusion framework may provide insights into enhancing the capabilities of our model.

Additionally, efforts can be made to optimize both the speed of image generation and the quality of the generated samples. This could involve refining the architecture of the model like introducing a ResNet block, exploring alternative training techniques, or incorporating additional contextual cues to increase the sample process's efficiency and accuracy. With the introduction of a ResNet block in future work, the model may be able to better capture more fine-grained details present in the CIFAR-10 dataset. Along with this, experimenting and obtaining a striking balance between the levels and steps per level may further increase the performance of the model.

In summary, future work should focus on further exploring the integration of contextual information, investigating different types of normalizing flows, and optimizing the speed and quality of image generation. By pursuing these directions, we can unlock new possibilities for improving the performance and applicability of our model in various domains. These ongoing endeavors will ensure that our model strikes a balance between efficient sampling time and high-quality output, paving the way for improved performance in future applications.

Bibliography

[1] Abdal, R., Zhu, P., Mitra, N. J., and Wonka, P. Styleflow: Attribute-conditioned exploration of stylegan-generated images using conditional continuous normalizing flows. *ACM Trans. Graph. 40*, 3 (may 2021).

[2] Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., and Levine, S. Stochastic variational video prediction, 2018.

[3] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020.

[4] Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. Wavegrad: Estimating gradients for waveform generation, 2020.

[5] Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., Dehak, N., and Chan, W. Wavegrad 2: Iterative refinement for text-to-speech synthesis, 2021.

[6] Croitoru, F.-A., Hondru, V., Ionescu, R. T., and Shah, M. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023), 1–20.

[7] Denton, E., Chintala, S., Szlam, A., and Fergus, R. Deep generative image models using a laplacian pyramid of adversarial networks, 2015.

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

[9] Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation, 2015.

[10] Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp, 2017.

[11] Dr. P.K., G. *Calculus-I.* Laxmi Publications Pvt Ltd, 2021.

[12] Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows, 2019.

[13] Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. nflows: normalizing flows in PyTorch, Nov. 2020. Available at `https://doi.org/10.5281/zenodo.4296287`.

[14] Fedus, W., Goodfellow, I., and Dai, A. M. Maskgan: Better text generation via filling in the ,2018.

[15] Ge, H., Xia, Y., Chen, X., Berry, R., and Wu, Y. Fictitious gan: Training gans with historical models, 2018.

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

[17] Gudovskiy, D., Ishizaka, S., and Kozuka, K. Cflow-ad: Real-time unsupervised anomaly detection with localization via conditional normalizing flows. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (January 2022), pp. 98–107.

[18] Hagemann, P. L., Hertrich, J., and Steidl, G. *Generalized Normalizing Flows via Markov Chains.* Elements in Non-local Data Interactions: Foundations and Applications. Cambridge University Press, 2023.

[19] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[20] Ho, J., Chan, W., Saharia, C., Whang, J., Gao, R., Gritsenko, A., Kingma,
D. P., Poole, B., Norouzi, M., Fleet, D. J., and Salimans, T. Imagen video: High definition video generation with diffusion models, 2022.

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

[22] Jaiswal, A., AbdAlmageed, W., Wu, Y., and Natarajan, P. Capsulegan: Genera-tive adversarial capsule network, 2018.

[23] Kawar, B., Elad, M., Ermon, S., and Song, J. Denoising diffusion restoration models. In *Advances in Neural Information Processing Systems* (2022), S. Koyejo, S. Mo-hamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., pp. 23593–23606.

[24] Kingma, D. P., and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolu-tions, 2018.

[25] Kobyzev, I., Prince, S. J., and Brubaker, M. A. Normalizing flows: An introduc-tion and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence 43*, 11 (nov 2021), 3964–3979.

[26] Kong, Z., Ping, W., Huang, J., Zhao, K., and Catanzaro, B. Diffwave: A versatile diffusion model for audio synthesis, 2021.

[27] Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research).

[28] Lam, M. W. Y., Wang, J., Huang, R., Su, D., and Yu, D. Bilateral denoising diffusion models, 2021.

[29] LeCun, Y., and Cortes, C. MNIST handwritten digit database.

[30] Lin, Z., Khetan, A., Fanti, G., and Oh, S. Pacgan: The power of two samples in generative adversarial networks, 2018.

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

[32] Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. Neural importance sampling, 2019.

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

[34] Odena, A., Olah, C., and Shlens, J. Conditional image synthesis with auxiliary classifier gans, 2017.

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

[36] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation, 2021.

[37] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models, 2022.

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

[39] Ryu, D., and Ye, J. C. Pyramidal denoising diffusion probabilistic models, 2022.

[40] Sajekar, S. D. Diffusion augmented flows: Combining normalizing flows and diffusion models for accurate latent space mapping, 2023. MSAI Thesis; under the guidance of Jaewoo Lee, University of Georgia.

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

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

[43] Sorkhei, M., Henter, G. E., and Kjellström, H. Full-glow: Fully conditional glow for more realistic image generation, 2021.

[44] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision, 2015.

[45] Wang, P. denoising-diffusion-pytorch, 2023. Available at `https://doi.org/10.5281/zenodo.4296287`.

[46] Zhang, Q., and Chen, Y. Diffusion normalizing flow, 2021.