Diffusion Augmented Flows: Combining Normalizing Flows and Diffusion Models for Accurate Latent Space Mapping

by

Soham Sajekar

(Under the direction of Jaewoo Lee)

Abstract

In this study, we propose a novel generative model that combines the strengths of two well-known generative models: normalizing flows and diffusion models. Normalizing flows lack the ability to fully map to Gaussian space, resulting in limited expressiveness. To overcome this, our model utilizes normalizing flows to map the complex data distribution to a latent distribution and then employs a diffusion model to make the latent distribution achieve equivalence to a Gaussian distribution. Additionally, we introduce a new training procedure that combines maximum likelihood estimation from normalizing flows and variational lower bound from diffusion models, resulting in a unified end-to-end architecture. We evaluate our model using the Fréchet Inception Distance and Negative Log-likelihood scores and show that our model outperforms Neural Spline Flows [7] and gives comparable results to traditional diffusion models [13]. Our work presents a promising direction in the field of generative modeling, specifically in image synthesis.

Index words: Generative models, Normalizing flows, Diffusion models, Fréchet Inception Distance, Negative Log-likelihood

Diffusion Augmented Flows: Combining Normalizing Flows and Diffusion Models for Accurate Latent Space Mapping

by

Soham Sajekar

B.Tech., Vishwakarma Institute of Technology, India, 2021

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

Masters of Science

Athens, Georgia

2023

Diffusion Augmented Flows:
Combining Normalizing Flows and Diffusion Models for Accurate Latent Space Mapping

by

Soham Sajekar

Approved:

Major Professors: Jaewoo Lee

Committee: Frederick Maier
Ninghao Liu

Electronic Version Approved:

Dean's Name Here
Dean of the Graduate School
The University of Georgia
August 2023

Acknowledgments

I would like to express my heartfelt gratitude to Prof. Jaewoo Lee, my major professor and thesis advisor, for his excellent assistance, unwavering support, and continual encouragement during my research journey. His vast expertise and insightful observations have considerably affected the direction and method of my study, molding it into its current shape. I am particularly thankful to the members of my thesis committee, Prof. Frederick Maier and Prof. Ninghao Liu, for their kind assistance and insightful input. Their skills and suggestions were critical to the effective completion of my project.

My greatest gratitude goes to my colleague, Sanika Katekar, for her invaluable support in numerous facets of my study. Her contributions were critical to the success of my research goals. Last but not least, I want to thank my cherished family and friends for their continuous support and aid during my academic endeavors. Their words of encouragement and physical presence have served as a continual source of motivation and inspiration.

I consider myself extremely lucky to have received the assistance and direction of these extraordinary people whose combined efforts made my research dreams a reality.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Generative models are a type of machine learning algorithm that creates new samples by learning the probability distribution of examples in a given dataset. Its ability to generate new realistic samples using various data types like images[32, 17], texts [38, 1], audio [19, 41], has increased their popularity in recent times. The main objective of generative models is to learn the underlying data distribution, and the ability to sample from the learned distribution is an added advantage of the generative models. These models have applications in numerous fields like robotics [35, 9], computer vision [40, 3], natural language processing [25], etc. Apart from direct applications in the real world, these models can also be used to further improve the existing machine learning models by providing fresh data which can be utilized for robust training and testing of models [2, 39]. This allows to produce more generalized models without losing accuracy. One very important application can be in the healthcare industry [37, 26, 28], where the data is very limited for the existing models.

In this thesis, our focus lies on an alternative category of generative models called normalizing flows [5, 6, 20, 12, 10] and diffusion models [13, 36]. These models represent a recent and actively evolving area of research.

Normalizing flows, also known as flow-based generative models, constitute a specific type

of generative model that uses invertible transformations. Their objective is to acquire information about a data distribution from a series of transformations that are both invertible and bijective. The primary aim is to establish a mapping between the original data distribution, which can be intricate and unknown, and a well-understood and simple distribution, such as the Gaussian distribution. By systematically applying a sequence of invertible transformations, normalizing flows empower us to explicitly model the data distribution. The central concept underlying normalizing flows involves initiating the process with a base distribution, typically a known distribution like the Gaussian distribution. Then, through the application of a sequence of learned transformations, the data is effectively mapped to the desired distribution. Importantly, each transformation is meticulously designed to be invertible, meaning that it possesses a well-defined inverse transformation. This property allows us to reliably map samples from the target distribution back to the base distribution or vice versa. These transformations are typically parametrized by neural networks, allowing for flexibility and expressiveness. By stacking multiple transformations together, a complex mapping can be learned, enabling the model to capture intricate structures and dependencies in the data. One of the notable advantages of normalizing flows is their ability to perform exact log-likelihood estimation. This means that they can provide an accurate measure of how likely a given sample is under the learned distribution. This property is valuable for tasks such as density estimation, where knowing the likelihood of a sample is essential. However, normalizing flows also has its limitations. One challenge is mapping complex data distributions to a pure Gaussian distribution. While normalizing flows aim to transform the data distribution into a known distribution, in practice, it is often observed that the assumption of the transformed data points following the base distribution is violated. This can lead to approximations and discrepancies when sampling from the inverse transformations.

Diffusion models approach the problem of learning the data distribution in a different manner. Instead of applying a series of transformations, diffusion models iteratively diffuse

noise over the actual data samples. The model then performs a series of steps, known as denoising steps, where it gradually transforms the noise sample to resemble a valid sample from the true data distribution. The number of timesteps used to diffuse noise determines the complexity of the modeling process. Diffusion models have the advantage of being able to generate highly realistic images. However, a downside is that diffusion models take a long time to train and sample as they require a larger number of timesteps. The larger number of timesteps ensures that the resulting noisy latent sample after adding noise to the actual sample is a valid sample from the base distribution.

To summarize, we have two distinct types of generative models known as normalizing flow models and diffusion models. These models are specifically designed to tackle the challenge of comprehending intricate data distributions. Normalizing flows accomplish this by employing a series of invertible transformations that effectively map the data distribution to a familiar and well-understood distribution. On the other hand, diffusion models adopt an iterative process where noise is gradually introduced into the data and subsequently learned to be removed or reduced. It is important to recognize that each approach possesses its own unique strengths and limitations, rendering them suitable for various scenarios and applications within the realm of generative modeling.

1.1 Approach

This paper addresses the limitations inherent in normalizing flows and diffusion models by introducing a novel generative model that leverages the strengths of both approaches. Our primary objective is to combine these two architectures to create a unified framework that overcomes the specific drawbacks associated with each model, ultimately achieving superior performance in generating high-quality samples.

The key motivation behind our work is to address the challenge faced by normalizing

flows in fully mapping complex data distributions to complete Gaussian distributions [31] [16]. To address this, our model applies a sequence of invertible transformations. These transformations aim to convert the original complex data distribution into a Gaussian distribution. However, it is important to note that even after these transformations, the resulting distribution may not perfectly adhere to a Gaussian distribution. To achieve a complete transformation to pure Gaussian distribution, we incorporate diffusion models into our architecture. The diffusion models play a vital role in further enhancing the intermediate distribution, ensuring a comprehensive transformation that captures the desirable characteristics of a Gaussian distribution. This integration allows us to harness the best of both worlds, mitigating the limitations of each model and achieving superior generative performance. For the sampling process, Gaussian noise is first passed through a denoising network which ensures the generation of latent samples following the diffusion model framework. Further, we pass these latent samples through a normalizing flow model, which uses its inverse transformations and knowledge of learned probability distribution to generate final samples from the latent samples. Through the sequential application of these two steps, we are able to generate high-quality images that accurately reflect the underlying data distribution.

To validate the efficacy of our proposed architecture, we conduct evaluations on three widely used datasets: MNIST [23], CIFAR-10 [22], and CELEB-A [24]. We employ two evaluation metrics such as Fréchet Inception Distance (FID) [11] [34] and Negative Log-likelihood. These metrics provide comprehensive insights into the performance of our model, allowing us to quantitatively demonstrate its superiority over existing approaches. We conducted experiments to compare our model with other models like Denoising Diffusion Probabilistic Models (DDPM) [13] and Neural Spline Flows (NSF) [7]. The experimental results demonstrate that our proposed method yields results comparable to those of DDPM while surpassing the performance of the NSF model. This indicates that our method holds the potential to significantly enhance existing normalizing flow models. Moreover, our model

exhibits lower negative log-likelihood values and FID scores compared to the NSF model, suggesting that our methodology achieves a superior quality of generated samples when compared to traditional normalizing flow models. On all three datasets, our model achieves FID scores comparable to those of DDPM, implying that the quality of the generated samples is on par with that of conventional diffusion models. Additionally, our model also has faster sampling times in comparison to diffusion models. By reiterating these findings, we underscore the ability of our proposed method to rival established approaches while offering improved accuracy and faster sampling times. This highlights the potential of our methodology to advance the field of generative modeling by combining the strengths of traditional diffusion models and normalizing flow models.

Chapter 2

Related Work

In recent years, normalizing flows and diffusion models have been widely studied in the generative modeling field of machine learning. This extensive research has led to many papers that propose various approaches to improve the performance of these models. Some of the important and interesting findings have been mentioned below, starting with comparatively basic types of generative models like Generative Adversarial Networks (GANs), Variational Auto-encoders (VAEs), and Autoregressive models:

GANs [8] are a type of generative model that have two parts, a generator network, and a discriminator network. The generator generates new fake samples by using the latent space mapping of the given distribution, and the discriminator tries to distinguish between these new fake samples with the original samples. The main objective of GANs is to minimize the discriminator loss while maximizing the generator loss. VAEs [21], on the other hand, encode the sample from the given distribution into a low dimensional latent space and then decode it back, reconstructing the new sample. The loss of VAEs consists of the reconstruction loss and KL divergence. Autoregressive models involve computing the joint probability of a sequence of attributes by calculating the conditional probabilities of each attribute in relation to its earlier attributes. These models provide an excellent way for estimating the original density

of the data distribution but lack in expressiveness; that is, the newly generated samples are not distinct enough from the actual ones.

Dinh et al. [6] proposed a generative model architecture (RealNVP) based on a series of affine transformations and non-linear coupling functions. The RealNVP model starts by splitting the input data sample into two counterparts and applies a series of invertible transformations only on one part, keeping the other part unchanged. This process is iteratively done along with adding perturbations in-between layers. This paper laid the main foundation for normalizing flows and other flow-based generative architectures.

Kingma et al. [20] introduced the use of invertible 1x1 convolutions in the flow-based generative models, enabling them to learn feature map transformations in a much more efficient manner. The key contribution of Glow lies in its use of invertible 1×1 convolutions, which are computationally efficient and allow for flexible modeling of dependencies between the input and output variables. These convolutions ensure that the transformation process is reversible, meaning that the original data can be accurately recovered from the generated samples. It leverages the concept of squeezing, which reduces the spatial dimensions of the data, making it easier to model. Additionally, the authors introduce a concept called multi-scale architecture, which enables the model to capture both global and local dependencies within the data. The basic idea behind the multi-scale architecture is to process the data at different scales. It involves dividing the input image into multiple levels, where each level represents a different scale of the image. At each level, the model learns to capture the dependencies specific to that scale while also considering the contextual information from other scales. In the Glow model, the multi-scale architecture is implemented using a sequence of levels. Each level consists of a squeeze transformation and act-norm layer along with two main operations: an invertible 1×1 convolution layer and a coupling layer. The coupling layer is responsible for splitting the input data into two parts and applying an element-wise affine transformation to one of the parts. This transformation allows the model

to capture local variations in the data. The other part remains unchanged and serves as a conditioning variable for the transformation. By applying this transformation, the model can model dependencies within the data at that scale. The invertible 1×1 convolution layer acts as a mixing operation that allows information to flow between different scales. It reshapes and reorganizes the data in a way that captures global dependencies. Importantly, this convolution is invertible, meaning that the original data can be accurately recovered from the transformed data. By stacking multiple levels, each with its coupling and convolution layers, the multi-scale architecture captures dependencies at different scales. The lower levels capture fine-grained details and local structures, while the higher levels capture more global patterns and larger-scale dependencies.

Grathwohl et al. [10] introduced a new type of flow-based generative model called Free-form Continuous Dynamics (FFJORD), which uses continuous-time differential equations to model the transformation of original data to a known distribution. Unlike other flow-based generative models, this model requires fewer parameters and computations to generate comparable results since they don't need to explicitly specify the number of transformations to map original data to a known distribution.

Durkan et al. [7] utilized a different type of transformation called piece-wise rational-quadratic spline transformation, as compared to the affine transformations in other baseline flow-based models. This enabled for modeling of high dimensional, complex, and multi-modal distribution, giving state-of-the-art results in the flow-based generative models. For our proposed architecture, we build upon the neural spline architecture combining it with the denoising diffusion probabilistic models (DDPM) proposed by Ho et al. [13], which is the main architecture that diffusion models are built. The diffusion process is where we make the model learn to denoise a pure noisy image which represents the Gaussian distribution. The DDPM architecture also involved the use of an attention mechanism and the incorporation of additional prior knowledge by conditionally training the model. Apart from generative

modeling, these models are also extensively used for other tasks such as image in-painting, text-to-image generation, etc.

DiffusionCNF [18] utilizes a conditional normalizing flow to model the diffusion process in diffusion models. DiffusionCNF aims to reduce the sampling time of the diffusion models by using conditional normalizing flow instead of the U-Net architecture that DDPM uses. DDPM uses a Gaussian assumption to denoise a sample in the reverse process. DiffusionCNF gets rid of this assumption by utilizing a normalizing flow conditioned on noisy samples, as normalizing flows have the ability to map complex data distributions to simple ones. While our approach aims to solve the drawbacks of normalizing flows, DiffussionCNF aims to solve the drawbacks of diffusion models.

Postels et al. [31] address the limitations of normalizing flows on real-world data. Real-world data usually exists on a lower-dimensional manifold within a higher-dimension space. Normalizing flows are good at learning the underlying data distribution, but they often seem to have problems when the data is in a lower dimensional surface called a manifold within a higher dimensional space. The paper mentions that finding the probability of a specific point on the manifold within a higher dimensional space becomes 0 for all points outside the manifold. Calculating the logarithm of the determinant of the Jacobian also becomes infinitely large. These issues cause an optimization problem to arise, leading to an ill-posed optimization problem. A lot of existing works solve this problem by adding noise to the manifold space [20, 15, 16]. The authors of [31] solve this problem by sampling directly from the original data distribution by using the knowledge of the noise model and the perturbed data distribution. They establish that normalizing flow models that are trained on the perturbed data implicitly capture the underlying manifold in regions where the data is most likely to be found. Using that information, they propose an optimization objective that finds the point in the higher dimensional space that is most likely closer to the point in manifold space. Horvat et al. [15] also aim to solve the same problem described by [31] by adding

noise to the manifold space. They proposed a methodology called Denoising Normalizing Flow (DNF) which follows a three-step process: Inflation, Learning, and Denoising. In the inflation phase, DNF adds noise to the manifold space in order to inflate it to higher dimensional data. This makes it easier to use normalizing flows to learn the distribution of data. Finally, DNF learns a denoising mapping similar to a denoising autoencoder. The loss function used in DNF calculates the error between the original data and denoised data.

Zhang et al. [42] have recently made efforts in the field of combining normalizing flows with diffusion models, proposing a novel architecture called "Diffusion Normalizing Flows." Their focus primarily revolves around the employment of the diffusion process, where normalizing flows are utilized to progressively transform noise-corrupted images. Their proposed solution effectively combines the strengths of normalizing flows and diffusion models, sharing similarities with our own proposed architecture. Nevertheless, it is crucial to acknowledge that there exist some distinctions between the two methodologies. In the case of "Diffusion Normalizing Flows," the overall process largely mirrors that of the diffusion process. Much like diffusion models, Diffusion Normalizing Flows encompass a forward process and a backward process. The forward process takes an input and subjects it to a function, incorporating some noise in the process. Conversely, the backward process entails a distinct transformation that encompasses both a drift term and an additional term. Notably, in Diffusion Normalizing Flows, the drift term within the backward process is amenable to learning, affording the model the capacity to adapt and enhance its performance over time. Essentially, Diffusion Normalizing Flows utilize these two processes to facilitate bidirectional data transformation, capitalizing on the ability to learn from and refine the transformation based on the unique characteristics of the data. In contrast, our architecture incorporates both normalizing flows and diffusion models within the learning process. This involves training and optimizing both the networks, that is, the CNN (Convolutional Neural Network) based flow network from normalizing flows models as well as U-Net from diffusion models.

Chapter 3

Preliminaries

This preliminaries section introduces some basic concepts and notations that are used throughout the paper. We start by defining distributions, invertible transformations, and the Jacobian as normalizing flows are inherently dependent on these concepts. Normalizing flow models are a class of generative models that learn to transform a simple, easy-to-sample distribution into a more complex distribution of interest. Diffusion models are a class of generative models that learn to generate data by gradually adding noise to a latent representation. Both these models will be explained in detail. Further, the mathematical concepts behind these models, the loss functions used to train these models, and the advantages and disadvantages of each of these will be discussed.

3.1 Notations

3.1.1 Distributions

A distribution can be thought of as a probability measure that assigns a probability to each possible outcome of a random experiment. The probability density function (pdf) associated with a distribution serves as a function that quantifies the likelihood of the random variable

assuming a specific value. To illustrate, the pdf of the standard normal distribution can be represented using the subsequent formula:

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \tag{3.1}$$

3.1.2 Invertible Transformations

An invertible transformation can be simply put as a mathematical function that can be reversed to get the original variable back from the transformed variable. In other words, if $f(x)$ is an invertible transformation, then there exists a function $f^{-1}(y)$ such that $f(f^{-1}(y)) = y$ for all y. For example, the function $f(x) = \log(x)$ is an invertible transformation, because its inverse is $f^{-1}(y) = \exp(y)$.

3.1.3 Jacobian

The Jacobian of a transformation $f(x)$ is a matrix that contains the partial derivatives of $f(x)$ with respect to x. The Jacobian can be used to compute the change of variables formula, which is a formula that relates the pdf of a transformed distribution to the pdf of the original distribution. The change of variables formula is given by the following equation:

$$p(x) = p(f(x)) \left| \frac{\partial f(x)}{\partial x} \right| \tag{3.2}$$

where $p(x)$ is the pdf of the original distribution, $p(f(x))$ is the pdf of the transformed distribution, and $|J(f(x))|$ is the absolute value of the determinant of the Jacobian matrix of $f(x)$.

3.1.4 Log-likelihood

The log-likelihood of a data point x can be expressed as the logarithm of the probability that x was generated from a distribution $p(x)$. This measure is commonly employed as a loss function in generative modeling. The log-likelihood of x is determined by the following equation:

$$\log p(x) = \log \left(\int p(x|y)p(y)dy \right) \tag{3.3}$$

Here, $p(x|y)$ denotes the likelihood of x given y, and $p(y)$ represents the prior distribution of y.

Throughout this document, we will utilize these symbols to explain normalizing flow models and diffusion models. As required, we will furnish supplementary clarifications regarding these symbols.

3.2 Normalizing Flows

Normalizing flows are generative models that map samples from a simple distribution to a target distribution of interest via a series of invertible transformations. Let $\mathbf{z}$ be a sample from a simple distribution, such as a standard Gaussian, and $\mathbf{x}$ be a sample from the target distribution $p_{\mathbf{X}}$. The goal is to learn a bijective mapping $f : \mathbf{z} \to \mathbf{x}$, where f can be represented as a composition of K invertible transformations $f_K \circ \cdots \circ f_1$, with $f_k : \mathbb{R}^D \to \mathbb{R}^D$.

Further, the probability density function (pdf) of $\mathbf{x}$ can be produced by modifying the pdf of $\mathbf{z}$, according to the change of variables formula, given $\mathbf{x} = f(\mathbf{z})$:

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(\mathbf{z}) \left| \det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right|^{-1} \tag{3.4}$$

It is possible to build a flow f that can accurately approximate the desired distribution

$p_{\mathbf{X}}$ by selecting the right transformations. A popular choice is the Affine transformation which is represented as follows:

$$f_k(\mathbf{z}) = \mathbf{u}_k \odot \mathbf{z} + \mathbf{b}_k \tag{3.5}$$

where $\mathbf{u}_k$ and $\mathbf{b}_k$ are learnable parameters, and $\odot$ represents the element-wise multiplication. The determinant of the f_k Jacobian matrix is calculated as follows:

$$\det \frac{\partial f_k(\mathbf{z})}{\partial \mathbf{z}} = \prod_{i=1}^{D} u_{k,i} \tag{3.6}$$

where $u_{k,i}$ is the i-th element of $\mathbf{u}_k$.

The parameters of the transformations are learned by maximizing the log-likelihood of the data during training:

$$\log p_\theta(\mathbf{x}) = \log p_{\mathbf{Z}}(\mathbf{z}) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right| \tag{3.7}$$

where θ denotes the parameters of the generative distribution $p_\theta(\mathbf{x}|\mathbf{z})$. The first term on the right-hand side can be easily computed if the simple distribution $p_{\mathbf{Z}}$ is known. The second term, which is the log-determinant of the Jacobian, can be efficiently computed using the LU decomposition.

3.2.1 Types of flows

Within the domain of normalizing flow models, the transformations applied to the data are referred to as flows. Two commonly utilized flow architectures are coupling flows and autoregressive flows, both of which offer expressive transformations for modeling intricate distributions. In the following section, we will delve into these flow types and provide specific examples.

3.2.1.a Coupling flows

Coupling flows are a form of transformation employed in numerous flow-based models to construct complex distributions. They operate by dividing the data into two segments and applying the transformation function solely to one segment while leaving the other untouched. Reversing the process allows us to convert the transformed data back into its original form. Models such as [6] and [5] use the coupling flows.

3.2.1.b Autoregressive flows

Autoregressive flows constitute another type of transformation that relies on autoregressive models. In such models, each data element is transformed based on preceding data elements. Consequently, the output at any given step is contingent upon the input at that step as well as all previous inputs. This transformation utilizes a set of functions that map the previous inputs to the parameters for each step. [29] is a popular work using autoregressive flows.

3.2.2 Coupling Functions

Coupling functions, mathematical functions employed within various flows, play a crucial role in coupling flows and autoregressive flows for data transformation. Coupling functions enable the introduction of intricate transformations to the data, allowing the capture of complex patterns. They can range from simple operations such as addition or multiplication by a constant to sophisticated approaches like neural networks. In the subsequent section, we will explore affine and spline coupling functions as they will be utilized in the proposed method.

3.2.2.a Affine coupling

Affine coupling is a type of coupling function commonly used in coupling flows. It provides a simple and efficient way to transform the data by adding or scaling the input with a set of parameters. Mathematically, the affine coupling can be defined as follows, with $x_{i:j}$ representing a subvector or subset of elements from a vector x. Specifically, it refers to the elements of x starting from index i up to (and including) index j:

$$y_{1:d} = x_{1:d} \tag{3.8}$$

The first d elements of y are the same as the first d elements of x.

$$y_{(d+1):D} = s(x_{1:d}; \theta) \odot x_{(d+1):D} + t(x_{1:d}; \theta) \tag{3.9}$$

The remaining $(D-d)$ elements of y are computed using the scaling factor $s()$ and translation factor $t()$.

In the above equations, d represents the number of dimensions in the input x, D represents the total number of dimensions, and $\odot$ denotes element-wise multiplication. The scaling factor $s(x_{1:d}; \theta)$ and translation factor $t(x_{1:d}; \theta)$ are functions that depend on the first d dimensions of x and the parameters θ. These functions can be implemented using neural networks or other mathematical functions to capture complex transformations.

Affine coupling is attractive because it allows for simple computation of the Jacobian determinant, which is essential for the flow's invertibility. Additionally, the coupling of variables ensures that each variable is influenced by the other variables in the flow.

3.2.2.b Spline coupling

Spline coupling functions utilize splines, which are piecewise-polynomial or piecewise-rational functions, as coupling functions. Splines offer flexibility in capturing complex transformations while ensuring invertibility. There are different types of spline coupling functions, and we'll focus on rational quadratic splines.

Rational Quadratic Splines: Rational quadratic splines are another form of spline coupling function. They are defined by specifying knots and derivatives at the inner points. These splines provide a flexible way to model the transformation [7].

Mathematically, a rational quadratic spline coupling function can be defined as follows, with $x_{i:j}$ representing a subvector or subset of elements from a vector x. Specifically, it refers to the elements of x starting from index i up to (and including) index j:

$$y_{1:d} = x_{1:d} \tag{3.10}$$

The first d elements of y are the same as the first d elements of x.

$$y_{(d+1):D} = r(x_{1:d}; \theta) \odot x_{(d+1):D} + s(x_{1:d}; \theta) \tag{3.11}$$

The remaining $(D-d)$ elements of y are computed using the rational function $r()$ and scaling factor $s()$.

In this equation, $r(x_{1:d}; \theta)$ and $s(x_{1:d}; \theta)$ are functions that depend on the first d dimensions of x and the parameters θ.

Overall, the coupling functions provide a way to achieve more flexible transformations in normalizing flows while maintaining invertibility. The choice of coupling function type depends on the specific requirements and characteristics of the data being transformed. For our proposed implementation, we utilized the rational quadratic spline coupling function as

it showed better performance.

3.3 Diffusion Models

Diffusion models are a dynamic type of generative model that has attracted a lot of attention lately and was inspired by non-equilibrium thermodynamics, providing a powerful framework for modeling complex probability distributions. The basic idea behind diffusion models is to start with a simple distribution, such as a uniform distribution, and then gradually add noise to the distribution over time. The fundamental idea is to replicate the process of noise addition that gradually changes a target distribution into a simple distribution using a Markov chain. The final diffusion sampling can be viewed as a continuous time-dependent transformation that converts the initial noise distribution into the desired distribution.

3.3.1 Forward process

The forward process in diffusion models involves iteratively adding noise to a given sample to generate a sequence of correlated samples. This is achieved using a Markov chain, where each step in the chain corresponds to a timestep. The timesteps in diffusion models represent discrete intervals or steps at which noise is added to the samples. These timesteps are predefined and determine the granularity of the diffusion process. The number of timesteps, denoted as T, is a hyperparameter set by the user. A higher number of timesteps allows for more precise modeling of the data distribution but increases computational complexity. At each timestep, noise is added to the current sample, leading to a new sample in the sequence. This process continues until the desired number of timesteps is reached. By iteratively applying this process, the initial sample x_0 is transformed into a sequence of correlated samples x_1, x_2, ..., x_T, where T is the total number of timesteps.

A Markov chain is defined as a sequence of random variables (or states) denoted as

$x_1, x_2, ..., x_T$, where T represents the number of steps or time points in the chain. Each variable or state x_t depends only on the previous state $x_t - 1$ and is independent of all other states in the chain. In other words, the future states in the Markov chain depend solely on the current state and not on the past history of the chain. This property is known as the Markov property. Mathematically, the Markov chain is defined by the conditional probability distribution of each state given the previous state, denoted as $P(x_t|x_{t-1})$. Let's denote the initial sample as x_0 and the variance of noise added at each timestep as β_t. The forward process can be expressed as [13]:

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}), \quad q(x_t|x_{t-1}) := \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right) \tag{3.12}$$

where x_t represents the sample at timestep t and β_t controls the magnitude of the noise added. This sequence captures the evolution of the data distribution from the data distribution to the target distribution.

3.3.2 Reverse process

The reverse process in diffusion models plays a crucial role in reconstructing the original sample from the sequence of correlated samples obtained through the forward process. By iteratively removing the added noise in reverse order, the diffusion model aims to recover the initial sample x_0.

Mathematically, the reverse process is defined as follows [13]:

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \quad p_\theta(x_{t-1} \mid x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t); \Sigma_\theta(x_t, t)) \tag{3.13}$$

In this equation, x_{t-1} represents the sample at timestep $t - 1$, which is obtained by subtracting the noise from the current sample x_t.

By applying this reverse process sequentially, starting from the final sample x_T and moving backward to the initial sample x_0, the diffusion model attempts to recover the original data sample. This reverse process effectively denoises the samples and can also fill in missing parts or in-paint the data.

3.3.3 Neural network architecture

To facilitate the learning of diffusion models, a neural network architecture is commonly employed. One popular choice is the U-Net architecture [33], which is well-suited for capturing the complex relationships within the data. The U-Net architecture consists of an encoder path and a decoder path.

In the context of diffusion models, the neural network architecture is designed to be conditioned on the timesteps. This means that the network takes the current timestep as an additional input and generates the corresponding transformation at each step of the diffusion process.

In addition to the architectural choice of the U-Net, there are specific considerations for predicting the noise in diffusion models. Instead of directly predicting the true data sample, the network is trained to predict the mean of the added noise. This approach aligns with [27] and its reparametrization technique. The reparametrization technique involves learning the distribution of the added noise and sampling from it to generate the noise values. By predicting the mean of the noise distribution instead of the true sample, the network effectively models the noise properties, which are essential for the reverse process and the denoising capabilities of the diffusion model.

3.3.4 Log-likelihood estimation

To train diffusion models and compute the training objective, we optimize the variational bound on the negative log-likelihood. This objective aims to maximize the negative log-likelihood of the data under the model while incorporating a divergence term to regularize the learning process. However, directly optimizing this objective can be challenging due to the intractability of the likelihood.

Let's denote the true data distribution as $p(x_0)$. The variational lower bound can be written as [13]:

$$\mathbb{E}\left[\log p(x_0)\right] = \mathbb{E}_q \left[\log p(x_T) - \sum_{t=1}^{T-1} \log p(x_{t-1}|x_t) q(x_t|x_{t-1}) \right] \quad =: L \tag{3.14}$$

However, optimizing this objective directly can be computationally expensive. Therefore, diffusion models leverage parametrization and Bayesian statistics and assume a specific form for the diffusion process to simplify the training objective.

Using these simplifications, the likelihood estimation problem can be reformulated as a mean squared error (MSE) problem. The preliminary training objective can be calculated as the MSE between the true data and the denoised data:

$$\mathbb{E}_{x_0,\epsilon} \left[\left| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t \right) \right|^2 \right] \tag{3.15}$$

Chapter 4

Methodology

This chapter delves into the comprehensive methodology to seamlessly integrate normalizing flows and diffusion model architectures for our proposed generative model. This section starts by explaining the limitations of current techniques. It moves on to present a thorough exploration of the implementation approach and algorithmic intricacies of the unified architecture, highlighting the intricacies of the forward process, backward process, and training procedure. Furthermore, it provides insights into specific implementation details, including the selection and utilization of distinct flow-based and diffusion models.

4.1 Limitations of current techniques

In normalizing flows, one of the challenges is to map complex data distributions to a complete normal distribution. This process involves transforming the input data through a series of invertible bijective transformations. The drawbacks of these bijective functions are that they struggle to efficiently compute the logarithm of their Jacobian determinant, making optimization challenging, and thus facing difficulties when dealing with random variables that exist on manifolds, leading to ill-posed optimization problems and loss of information

about the original distribution [31]. As a result, when sampling from the learned distribution using the inverse transformations, certain approximations are introduced.

To elaborate on this, let's consider a normalizing flow with a series of invertible transformations parametrized by θ, denoted as f_θ. The goal is to learn a mapping from a complex data distribution p_{data} to a simple base distribution, typically a multivariate normal distribution p_z.

During the forward pass of the normalizing flow, the input data x is transformed through a sequence of functions f_θ: $z = f_\theta(x)$. Each transformation f_θ introduces a change of variables, which requires computing the determinant of the Jacobian matrix. The Jacobian determinant captures the effect of the transformation on the probability density function (PDF) of the data. The inverse pass of the normalizing flow involves sampling from the learned distribution by applying the inverse transformations. Given a sample z from the base distribution, the inverse transformations g_θ attempt to map it back to the original data space: $x = g_\theta(z)$

However, due to the complexity of the data distribution, the learned normalizing flow may not perfectly approximate the true data distribution. This discrepancy between the true data distribution p_{data} and the approximated distribution p_θ introduces certain deviations during the inverse transformation process. The sampling process assumes z, the output of forward transformation, follows Gaussian distribution, but in practice, it often largely deviates from the Gaussian distribution, which in turn causes $g_\theta(z)$ to deviate from the data distribution. This can result in sample data points that do not perfectly align with the true data distribution and may exhibit certain deviations or imperfections.

Overall, while normalizing flows offer a powerful framework for learning complex data distributions, the approximations involved in mapping to a complete normal distribution can introduce certain imperfections when sampling using inverse transformations. To solve this problem of the normalizing flow, we add a diffusion model at the end of the normalizing flow

to ensure that the latent representation reaches pure Gaussian distribution. The output of the normalizing flow is passed as an input of the diffusion model. The forward process of the diffusion model takes the output of the normalizing flow as the input and progressively adds noise to transform it into a complete Gaussian.

4.2 Algorithm

Algorithm 1: Training

repeat

$x \in D$;

$t \leftarrow \text{Uniform}(\{1, 2, ..., T\})$;

$\epsilon \sim \mathcal{N}(0, I)$;

$z_0 \leftarrow f_\theta(x)$, where

$f_\theta = f_1 \circ \ldots \circ f_{N-1} \circ f_N$;

$z_T = \sqrt{\bar{\alpha}_t} z_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$,

Take the gradient descent step on:

$\mathcal{L}diff = \mathcal{L}simple + \lambda * \mathcal{L}vlb$

$\mathcal{L} = \mathcal{L}diff + \log \left| \det \frac{\partial f_\theta(x)}{\partial x^T} \right|$;

until *converged*;

Algorithm 2: Sampling

$z_T \sim \mathcal{N}(0, I)$;

for $t = T$ **to** 1 **do**

$n \sim \mathcal{N}(0, I)$, if $t > 1$, else $n = 0$;

$z_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(z_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(z_t, t) \right) + \sigma_t n$;

$x = f_\theta^{-1}(z_0)$, where $f_\theta^{-1} = f_N^{-1} \circ f_{N-1}^{-1} \circ \ldots \circ f_1^{-1}$;

Output: x

Figure 4.1: Training and Sampling Algorithms

The forward process within our proposed architecture starts with an original data sample denoted by x, where $x \in D$. This sample is then subjected to a series of transformation layers within the normalizing flow section, resulting in the generation of a transformed variable, z_0. The series of transformation layers can be mathematically viewed as a composition of various bijective functions given by:

$$f_\theta = f_1 \circ \ldots \circ f_{N1} \circ f_N \tag{4.1}$$

The transformed variable z_0 is generated by passing x through the composition of bijective functions f_θ and is given by:

$$z_0 = f_\theta(x) \tag{4.2}$$

Significantly, this transformed variable represents a sample extracted from an intermediate simple latent distribution. Subsequently, the transformed variable z_0 is provided as an input to the diffusion part of the model, where an additional noise component is incorporated. This noise injection process facilitates the production of a pure noise variable, denoted as z_T, which is assumed to follow a Gaussian. z_T is obtained using the closed-form solution of the diffusion model.

$$z_T = \sqrt{\alpha_t} z_0 + \sqrt{1 - \alpha_t}\epsilon, \tag{4.3}$$

where $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \frac{1}{t}\sum_{s=1}^{t} \alpha_s$. β is the variance schedule that is used to add noise to the data set, z_0 is the transformed variable generated by a set of bijective transformations, and ϵ is noise. The integration of the diffusion model after the normalizing flow brings notable advantages, primarily addressing the limitation of the normalizing flow's inability to fully map to a Gaussian distribution. By leveraging the closed-form solution offered by the diffusion model in the forward process, a complete mapping to Gaussian noise is achieved.

On the other hand, the backward process commences with the sampling of a random noise vector, z_0, drawn from a Gaussian distribution. This noise vector is then subjected to the sampling process of the diffusion model, resulting in the generation of a transformed variable, z. This transformed variable serves as an intermediate sample originating from the simple latent distribution. Finally, the transformed variable z undergoes a series of inverted normalizing flow layers, ultimately yielding the final output, x, which represents the generated samples.

The incorporation of the diffusion model after the normalizing flow yields distinct advan-

tages. By first denoising a complete Gaussian noise z_T to an intermediate noisy sample z_0 using fewer diffusion steps, the diffusion model ensures that the subsequent passage of this noisy sample through the normalizing flow model facilitates the generation of high-quality final samples x.

The training procedure for our proposed architecture entails the optimization of both the normalizing flow layers and the diffusion models. The normalizing flow layers are trained using maximum likelihood estimation, wherein the loss function incorporates the log-likelihood of the transformed variable and the determinant of the Jacobian matrix. Concurrently, the diffusion model undergoes training utilizing a hybrid approach that combines the variational lower bound and mean squared error (MSE) loss function, as proposed in [27]. The primary objective of the training procedure is to minimize the combined loss function, which consolidates the loss contributions from both models.

In the combined architecture, the log-likelihood loss of the normalizing flows is replaced by the diffusion loss, facilitating the unification of the loss from both models. Consequently, the combined loss function is defined as the sum of the diffusion loss and the normalizing flow. This formulation ensures a comprehensive optimization process that harnesses the strengths of both normalizing flows and diffusion models.

We use maximum likelihood estimation to train the normalizing flow layers in the combined architecture:

$$\mathcal{L}_{nf} = \log(p_Z(f_\theta(x))) + \log \left| \det \frac{\partial f_\theta(x)}{\partial x^T} \right| \tag{4.4}$$

And the hybrid of the variational lower bound and MSE loss function to train the diffusion model part of the combined architecture:

$$\mathcal{L}_{diff} = |\boldsymbol{\epsilon} - \boldsymbol{\epsilon}\theta(\bar{\alpha}_t, f_\theta(x) + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}, t)|_2^2 \tag{4.5}$$

For a unified architecture, we combine the loss of both models by replacing the $\log(p_Z(f(x)))$ from the log-likelihood loss of normalizing flows with the diffusion loss. Then, the combined loss function is defined as:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}diff + \log \left| \det \frac{\partial f_\theta(x)}{\partial x^T} \right| \\ \mathcal{L} &= |\boldsymbol{\epsilon} - \boldsymbol{\epsilon}\theta(\bar{\alpha}_t, f_\theta(x) + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}, t)|_2^2 + \log \left| \det \frac{\partial f_\theta(x)}{\partial x^T} \right| \end{aligned} \tag{4.6}$$

By meticulously integrating the forward process, backward process, and training procedure, our proposed architecture harnesses the power of normalizing flows and diffusion models, leading to the generation of high-quality samples that capture the underlying data distribution. The utilization of this methodology sets the stage for subsequent chapters, allowing for detailed experimentation, analysis, and conclusive findings.

4.3 Implementation Details

In terms of implementation, we first start with the implementation of the normalizing flow part of the architecture. We implement a popular normalizing flow architecture such as Spline Flows [14] [7], which uses the Glow [20] as the first part of our architecture. Next, we implement the diffusion model part of our architecture. We use Denoising Diffusion Probabilistic Models (DDPM) [13] for the diffusion process. We also use a suitable optimizer, such as Adam, to minimize the loss function for training the normalizing flow model and the diffusion model. Finally, we combine the normalizing flow and diffusion model into a unified end-to-end architecture using our proposed training procedure.

Figure 4.2 shows the overall forward process of mapping the original data to Gaussian distribution. We can see that the Normalizing Flows on its own cannot completely transform the original data to pure Gaussian noise.

Overall, our implementation involves carefully selecting hyperparameters such as learning

rate, batch size, and number of layers for both the normalizing flow and diffusion model parts of the architecture. We will also use techniques such as early stopping and learning rate annealing to prevent overfitting and ensure the stability of the training process.

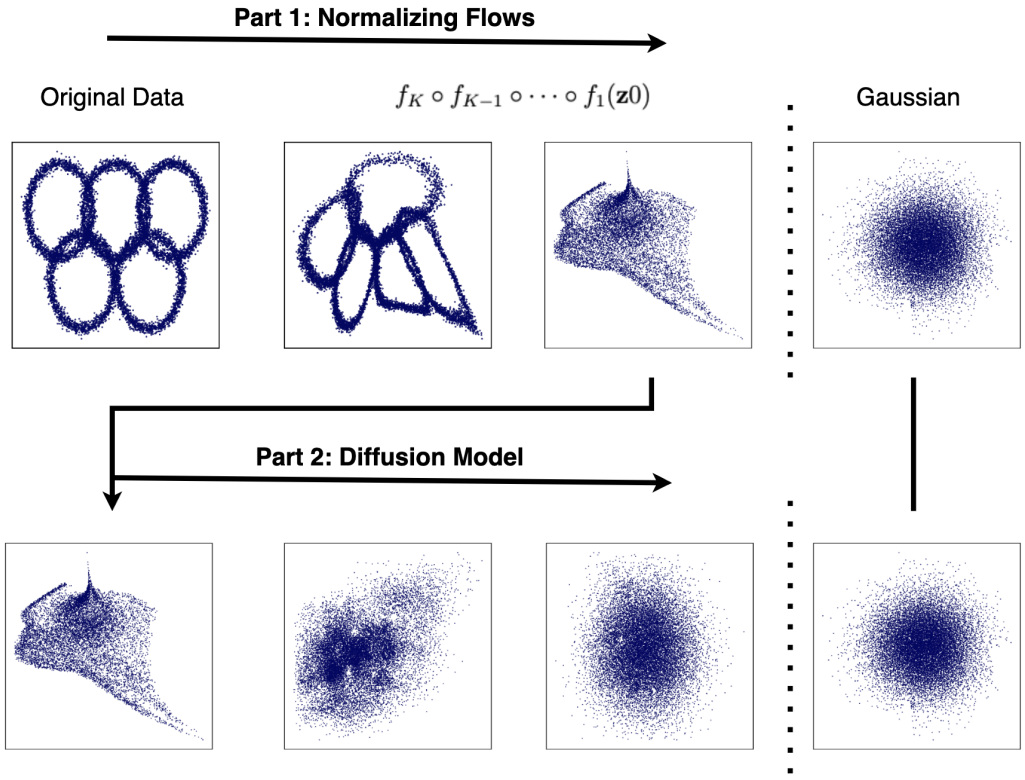

Figure 4.2: Forward process of the proposed architecture. Part 1: Using normalizing flows to map original data to a simpler latent distribution. Part 2: Further, using diffusion models to completely transform the intermediate samples to pure Gaussian noise

Chapter 5

Experiments

The experiments conducted in this thesis aim to evaluate the performance of our proposed model and compare it with existing models in the field, such as Denoising Diffusion Probabilistic Models (DDPM) [13] and Neural Spline Flows (NSF) [7]. To achieve this, we have selected well-established datasets such as MNIST[23], CIFAR10[22], and CELEBA[24]. These datasets have been widely used in the academic setting, providing a standardized benchmark for evaluating the capabilities of various models. In this chapter, we will describe the experimental setup, including the environmental setup, evaluation metrics, and training configurations, and present the obtained results. By conducting these experiments, we seek to gain insights into the effectiveness and potential of our model, contributing to the advancements in the field of generative models.

5.1 Environmental Setup

For running experiments on our model, the systems employed for conducting the research played a crucial role. The primary computing resource utilized were various computers available in the Artificial Intelligence lab of the University of Georgia, which featured a powerful

and dedicated NVIDIA's RTX A5000 Graphics Processing Unit (GPU). The system boasted notable specifications, including a substantial amount of Random Access Memory (RAM) of 256 Gigabytes (GB), ensuring efficient handling of large-scale computational tasks. The experiments were conducted using the PyTorch framework [30], which is widely recognized for its extensive capabilities in deep learning research. Specifically, version 1.13.1 was employed to facilitate the implementation of various neural network models and algorithms. Furthermore, the experiments relied on using Compute Unified Device Architecture (CUDA) [4] with version V11.6.124 for accelerated GPU computing, enhancing the performance and speed of the computations. CUDA is a parallel computing platform developed by NVIDIA for utilizing the power of NVIDIA GPUs for general-purpose computing. The Python programming language, in its Python 3.8.16 iteration, served as the primary coding language, providing a versatile and user-friendly environment for developing and executing the experimental procedures. Collectively, these meticulously chosen system specifications, along with the PyTorch framework and specific versions of CUDA and Python, formed the foundation for conducting the experiments in a robust and efficient manner.

5.2 Evaluation Metrics

In the evaluation metrics section, we utilize two crucial metrics to evaluate the performance of generative models: Negative Log-Likelihood (Nll) and Fréchet Inception Distance (FID). The negative log-likelihood metric measures the generative model's ability to accurately represent the underlying distribution of the training data. A lower value for negative log-likelihood indicates superior model performance. Conversely, FID is a widely acknowledged metric that assesses the similarity between the distributions of generated and real images. Together, these two metrics provide a comprehensive evaluation of generative models, offering valuable insights into their capabilities.

5.2.1 Fréchet Inception Distance (FID)

The Fréchet Inception Distance (FID) [11] [34] has gained widespread recognition as an essential evaluation metric utilized in the assessment of generated images within the domain of generative models. This metric offers a robust measure of similarity between the distribution of generated images and that of real images. By comparing the features extracted from the generated and reference images using a pre-trained Inception network, FID quantifies the distance between the two distributions. Lower FID scores correspond to higher quality and increased diversity among the generated images, indicating a closer resemblance to the distribution of real images [11]. FID scores are dependent on the dimension of the feature vectors chosen. For the purpose of experimentation, we chose the feature vector dimension to be 192 because this selection ensured a separation between scores which led to efficient comparison. This is an important point to consider before comparing our model with scores in other papers.

For the purpose of evaluating the MNIST dataset, a custom-trained MNIST classifier was utilized in place of the inception network typically employed in FID calculations. This modification proved advantageous as it facilitated the efficient conversion of MNIST samples into feature vectors, which were subsequently utilized for FID evaluation. This approach ensured seamless integration of the FID evaluation process with the unique characteristics of the MNIST dataset, allowing for comprehensive analysis and assessment of the quality and diversity of generated MNIST images.

5.3 Training Configurations

The training configuration for the thesis experiments involves several parameters that have a significant impact on the performance of the model. In this section, we will discuss these parameters and their effects on the MNIST, CIFAR10, and CELEBA datasets. The training

configurations are summarized in Table 5.1.

Parameters	**MNIST**	**CIFAR10**	**CELEBA**
Epochs	300	500	100
Batch Size	128	156	68
Image Size	(1, 32, 32)	(3, 32, 32)	(3, 64, 64)
Levels	3	4	4
Steps per levels	8	15	10
Hidden Channels	64	128	128
Resnet blocks	2(nflow); 2(dpm)	3(nflow); 2(dpm)	3(nflow); 2(dpm)
Timesteps	500	500	500

Table 5.1: Training Configurations

Let's begin by discussing the parameters related to the normalizing flow part of the proposed model, which employs the Glow architecture from [20], specifically the levels and steps per level parameters. The levels parameter represents the number of iterations in the multi-scale architecture of the Glow framework. For the MNIST dataset, 3 levels are utilized, while CIFAR10 and CELEBA employ 4 levels. At each level, the input data undergoes a squeeze transformation which reduces the dimensionality of the data allowing the network to learn from multiple latent dimensions. Increasing the number of levels allows the model to capture complex patterns at different scales. Each level consists of a squeeze transformation, an act-norm layer, and an invertible 1x1 convolution layer followed by steps per level. The steps per level parameter signifies the number of coupling transformations applied at each level. In the experiments, the MNIST dataset employs 8 steps per level, CIFAR10 uses 15 steps per level, and CELEBA utilizes 10 steps per level. Increasing the number of steps per level allows the model to better approximate complex data distributions into base distributions like Gaussian. This also makes the model more expressive, and it is able to generate high-quality samples. The combination of levels and steps per level parameter determines the total number of transformations or flows used in the normalizing flow part of the proposed model. Thus for MNIST, CIFAR10, and CELEBA, we used a total of 24, 60, and 40

transformations, respectively. Overall we used less number of levels and steps per level for the MNIST dataset as compared to the other datasets since the dataset has a single channel and consists of simple handwritten digits.

Other parameters include hidden channels and resnet blocks which respectively determine the number of channels or dimensions used in the hidden layers and the number of residual blocks used in both the flow network from the normalizing flow part and U-Net from the diffusion part of the model. The residual blocks used in the flow network are denoted by "nflow," and that used in the U-Net are denoted by "dpm". The MNIST dataset employs 64 hidden channels with 2 resnet blocks for both the networks, nflow and dpm, while CIFAR10 and CELEBA use 128 hidden channels with 3 resnet blocks in the nflow network and 2 resnet blocks in the dpm network. Increasing the number of hidden channels and resnet blocks enhances the model's ability to capture intricate features but also increases computational complexity. Again, we kept the number of hidden channels and resnet blocks less for the MNIST model as compared to the models. For MNIST, the model was trained for 300 epochs with a batch size of 128, CIFAR10 for 500 epochs with a batch size of 156, and CELEBA for 100 epochs with a batch size of 68. The number of epochs and batch size were determined considering the time and computational constraints available. Another crucial parameter is the timesteps, which determine the number of steps used for adding noise in the diffusion part of our proposed model. Increasing the number of timesteps allows the diffusion process to more accurately model complex data distributions. The experiments use 500 timesteps for all datasets. Mainly in the DDPM model [13], the experiments were performed using 1000 timesteps. Thus, we wanted to demonstrate that our proposed model can utilize a lesser number of timesteps and still generate comparable samples to the DDPM model. Careful selection of these parameters, as shown in Table 5.1, is essential to achieve optimal performance in the trained models. The choices made in the table consider dataset characteristics, model complexity, and computational constraints. By appropriately tuning

these parameters, the models can effectively learn from the data and generate high-quality outputs.

5.4 Results

This section presents a thorough analysis of the experimental findings obtained from the MNIST, CIFAR10, and CELEBA datasets. We provide a comprehensive evaluation of our model's performance, focusing on key metrics such as the Negative Log-Likelihood (Nll) and Fréchet Inception Distance (FID). Furthermore, we conduct a comparative analysis, benchmarking our model against established DDPM [13] and NSF [7] models. For model comparison, we evaluated the models on MNIST and CELEBA datasets. We specifically choose MNIST and CELEBA datasets because generative models can pose difficulties in evaluating and comparing solely through quantitative results. Using MNIST and CELEBA datasets, we can visually check the generated samples and give conclusions about the models. This comparison allows us to identify the unique strengths and limitations of our proposed approach, showcasing its potential contributions to the field of generative modeling. The evaluation results on different datasets are summarized in Table 5.2.

Table 5.2: Evaluation on Datasets

Datasets	**NLL**	**FID**
MNIST	0.93	1.85
CIFAR10	2.53	11.22
CELEBA	0.58	6.94

In relation to the MNIST dataset, our generative model demonstrates remarkable performance with an NLL of 0.93 and an FID of 1.85. These values indicate the model's capability to effectively capture the underlying distribution of MNIST data. A lower FID score implies a noteworthy similarity between the generated samples and authentic images. The generated samples can be observed in Figure 5.1, clearly exhibiting the model's precise

generation of new MNIST samples. This exceptional fidelity can be attributed primarily to the employment of an adequate number of flows and timesteps for modeling the MNIST data. Additionally, the simplicity and distinctive nature of the grayscale MNIST dataset contribute to the model's proficiency in achieving high-quality results.

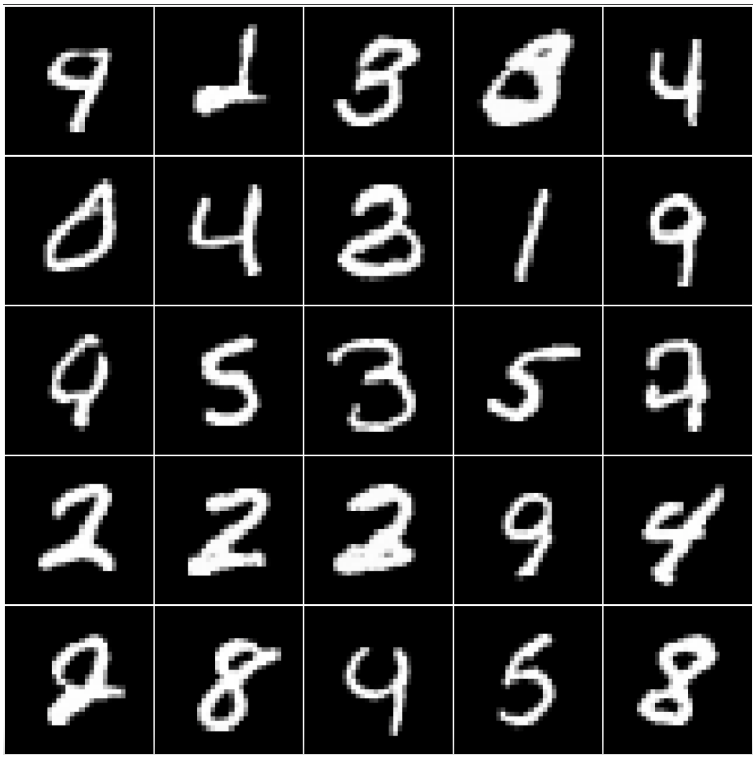

Figure 5.1: Samples generated using the MNIST dataset

On the CIFAR10 dataset, our model exhibits an NLL of 2.53 and an FID score of 11.22, as seen in Table 5.2. The generated samples for the CIFAR10 dataset can be seen in Figure 5.3. FID score of 11.22 is higher as compared to other datasets, demonstrating poor modeling performance on the CIFAR10 dataset. This can be mainly because the CIFAR10 data usually require a higher number of training epochs to converge properly. This can be seen in Figure 5.2, which displays the training loss (NLL) curve for the CIFAR10 dataset. It is observed that the model hasn't converged even after 500 epochs. However, it is not certain if more epochs and better loss convergence can improve the model performance significantly.

Another reason for the high FID score can be attributed to the high number of levels and dimensions of the CIFAR10 images. Since the CIFAR10 images are 32x32 pixels, employing a higher number of levels in our model can lead to excessive compression of the image data. Each level in our model utilizes a squeeze transformation, which applies a down-sampling operation. With each squeeze operation employing a squeeze factor of 2, after utilizing four levels, the final data dimension reduces to 2x2. This significant reduction in dimensionality can result in the loss of fine details and intricate structures present in the CIFAR10 images. Consequently, the generated samples may struggle to accurately capture the complex characteristics of the dataset. To address this issue, future research should focus on optimizing the level configuration of the model and finding a balance between dimensionality reduction and preserving important image details.

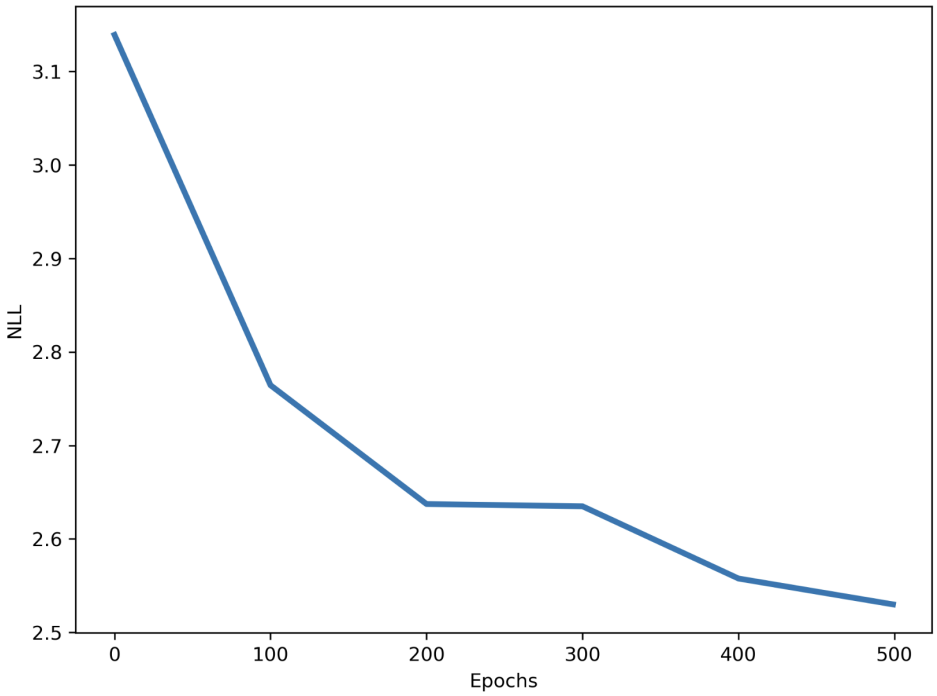

Figure 5.2: CIFAR10 Training Loss (NLL vs Epochs)

For the CELEBA dataset, our generative model achieves an NLL of 0.58 and FID score of 6.94; check Table 5.2 for evaluation metrics and Figure 5.4 for generated samples. It

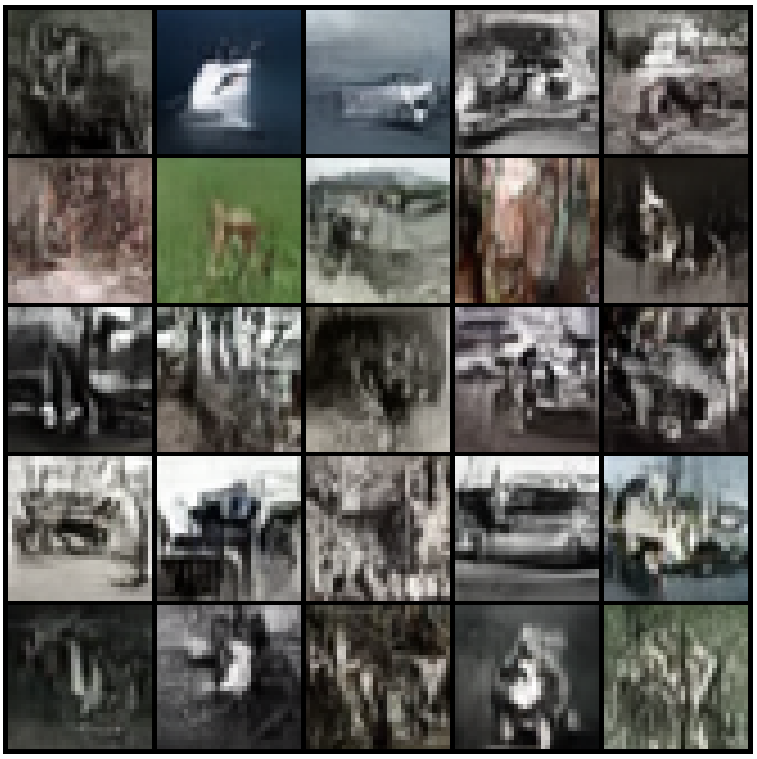

Figure 5.3: Samples generated using the CIFAR10 dataset

is observed that the generated samples are successfully able to generate human faces even though the FID score is moderately high. Sometimes, evaluation metrics like FID fail to capture the qualitative component of images, due to which the scores can be high, although the generated samples can show good results. Our model was able to capture the overall distribution of facial features in the CELEBA dataset, generating realistic images of human faces. One notable observation was made during the analysis of the generated samples from the CELEBA dataset. It was observed that all the generated samples exhibited a grey color background, see Figure 5.4. One possible explanation for the consistent grey background lies in the architectural choice of our model, where we implemented the glow architecture. Strikingly, in the original Glow paper [20], similar results were observed with generated images having a white or grey background for the CELEBA dataset. This correspondence

implies that the architectural characteristics specific to the glow model could be influential in shaping the uniformity of background colors. Additionally, we utilized dequantization as a preprocessing step in our model, a common practice to convert discrete pixel values into continuous representations. However, the precise effects of this preprocessing step on the color distribution, including the background, necessitate further investigation and detailed analysis.

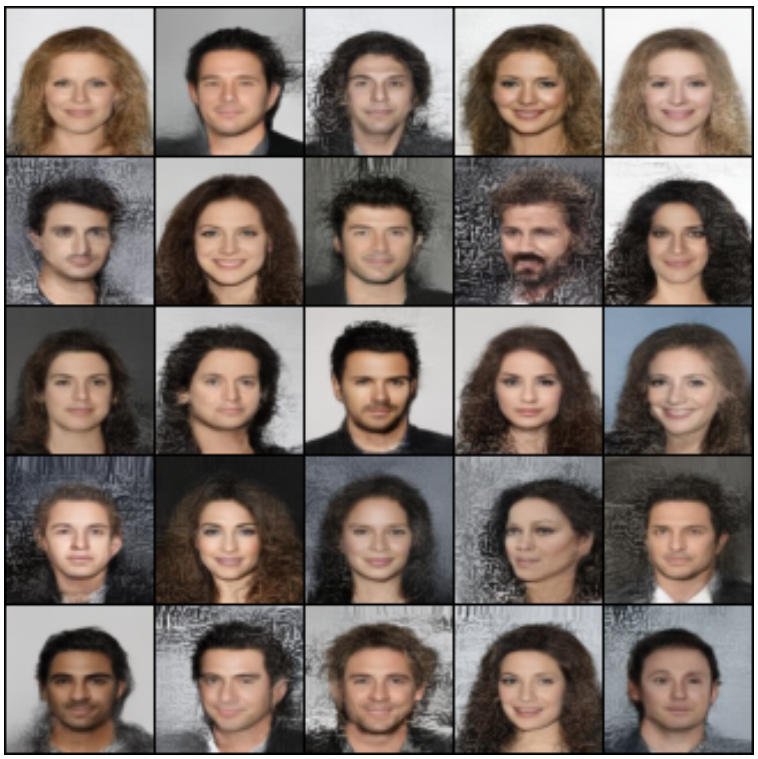

Figure 5.4: Samples generated using the CELEBA dataset

5.4.1 Model Comparison

We conducted a comprehensive analysis and comparison of our proposed model, Diffused-Flows, with two established models, DDPM and NSF. The results, as shown in Tables 5.3 and 5.4, highlight the performance of each model on the CELEBA and MNIST datasets, respectively, considering factors such as loss, FID score, and sampling time. For the CELEBA

dataset, the DDPM model and the NSF model both underwent training for 100 epochs with 1000 timesteps for the DDPM model and 80 flows for the NSF model, while our proposed Diffused-Flows model employed a combination of 500 timesteps and 40 flows. Likewise, for the MNIST dataset, the DDPM model utilized 1000 timesteps, and the NSF model utilized 48 flows and were both trained for 300 epochs, while our model incorporated a combination of 500 timesteps and 24 flows with 300 epochs.

Notably, despite our model using fewer timesteps and flows compared to DDPM and NSF, it achieved comparable results to DDPM and even outperformed the NSF model. By emphasizing these details, we underscore the efficiency and effectiveness of our proposed model in generating high-quality samples, showcasing its potential as a resource-efficient and practical approach in the field of generative modeling. It is important to note that a direct comparison of loss values between the DDPM model (MSE loss) and the other models (NLL loss) may not be appropriate since they use different loss measures. We can still draw insights based on the relative performance of the models in terms of FID scores and sampling times.

Table 5.3: Model Comparison on CELEBA

Models	**Loss**	**FID**	**Sampling Time (secs)**
DDPM	0.033 (MSE)	3.59	46.02
NSF	2.86 (NLL)	8.24	0.23
Diffused-Flows *(our model)*	0.58 (NLL)	6.94	22.6

For the CELEBA dataset, the DDPM model achieved a loss of 0.033 (measured in MSE), an FID score of 3.59, and a sampling time of 46.02 seconds for generating 25 samples. On the other hand, the NSF model exhibited a loss of 2.86 (measured in NLL), an FID score of 8.24, and a significantly shorter sampling time of 0.23 seconds. In comparison, our proposed Diffused-Flows model showcased competitive performance with a loss of 0.58 (measured in NLL), an FID score of 6.94, and a sampling time of 22.6 seconds. Check Figure 5.5 for

Figure 5.5: Model Comparison: generated samples on CELEBA

generated samples.

Table 5.4: Model Comparison on MNIST

Models	**Loss**	**FID**	**Sampling Time (secs)**
DDPM	0.029 (MSE)	1.05	28.97
NSF	1.03 (NLL)	2.58	0.21
Diffused-Flows *(our model)*	0.93 (NLL)	1.85	15.12

On the MNIST dataset, the DDPM model achieved a loss of 0.029 (measured in MSE), an FID score of 1.05, and a sampling time of 28.97 seconds. Similarly, the NSF model demonstrated a loss of 1.03 (measured in NLL), an FID score of 2.58, and a faster sampling time of 0.21 seconds. In this scenario, our proposed Diffused-Flows model again displayed competitive performance, achieving a loss of 0.93 (measured in NLL), an FID score of 1.85, and a sampling time of 15.12 seconds. Check Figure 5.6 for generated samples.

Considering FID scores across both datasets, our Diffused-Flows model performed competitively, exhibiting FID scores closer to those of the DDPM model and outperforming the NSF model. These results imply that our model successfully captured the essential characteristics of the datasets, generating samples with high fidelity. Sampling time is an important

Figure 5.6: Model Comparison: generated samples on MNIST

factor to consider for practical applications. In this regard, our Diffused-Flows model struck a balance between the DDPM and NSF models. It demonstrated faster sampling times compared to DDPM, making it more suitable for generating samples within a reasonable timeframe. While the NSF model had the fastest sampling time, it came at the expense of slightly higher FID scores, indicating a potential trade-off between speed and sample quality. To summarize, the results from our model comparison reveal that our proposed Diffused-Flows model performed competitively with both DDPM and NSF models on the CELEBA and MNIST datasets. It achieved comparable or lower FID scores while maintaining a favorable sampling time, thereby striking a balance between fidelity and efficiency. These findings underscore the potential of our model as a promising generative model with the ability to produce high-quality samples in a timely manner, making it suitable for various applications in the field of generative modeling.

Chapter 6

Conclusion

In this thesis, we proposed a novel generative model combining normalizing flows and diffusion models, which mainly helped in overcoming the limitations of latent space mapping in normalizing flows. The main contribution of this work is to allow the modeling of complex data distributions by integrating normalizing flows and diffusion models, which helps to incorporate the advantages of both architectures, such as the invertibility and quick sampling of normalizing flows and efficient mapping of complex distribution to known distribution like standard normal diffusion models of diffusion models.

Our proposed method has the limitation of high model complexity. Since we use the U-Net in the diffusion process as well as the convolutional neural network-based transformation networks for creating flows in normalizing flows, the overall model complexity can pose issues for computational resources. Further, this concept can be improved by coming up with different ways of combining these two models, where we closely observe the limitations available and find ways to overcome these limitations; for example, decreasing sampling time in diffusion models by replacing or slimming down the existing U-Net network with another network. Another future direction can be to use a pre-trained diffusion model, which can help in reducing the overall model complexity.

In conclusion, our proposed method provides a new direction in the future of generative modeling, where the integration of various models can help achieve high-quality results in a wide range of applications in the machine learning domain, such as computer vision, natural language processing, etc.

Bibliography

[1] Ahn, H., Ha, T., Choi, Y., Yoo, H., and Oh, S. Text2action: Generative adversarial synthesis from language to action. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 5915–5920.

[2] Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis, 2019.

[3] Cao, Y.-J., Jia, L.-L., Chen, Y.-X., Lin, N., Yang, C., Zhang, B., Liu, Z., Li, X.-X., and Dai, H.-H. Recent advances of generative adversarial networks in computer vision. *IEEE Access 7* (2019), 14985–15006.

[4] Cook, S. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, 1st ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2012.

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

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

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

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

[9] Gothoskar, N., Lázaro-Gredilla, M., Agarwal, A., Bekiroglu, Y., and George, D. Learning a generative model for robot control using visual feedback, 2020.

[10] Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018.

[11] 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.

[12] Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design, 2019.

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

[14] Ho, J., Jain, A., and Abbeel, P. Denoising Diffusion Probabilistic Models: diffusion models (ddpm) in PyTorch, 2023. Available at `https://github.com/bayesiains/nsf`.

[15] Horvat, C., and Pfister, J.-P. Denoising normalizing flow. In *Advances in Neural Information Processing Systems* (2021), M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., pp. 9099–9111.

[16] Horvat, C., and Pfister, J.-P. Density estimation on low-dimensional manifolds: an inflation-deflation approach, 2022.

[17] Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks, 2019.

[18] Katekar, S. S. Diffusioncnf: Learning denoising diffusion models via conditional normalizing flows, 2023. MSAI Thesis; under the guidance of Jaewoo Lee; University of Georgia.

[19] Kim, S., gil Lee, S., Song, J., Kim, J., and Yoon, S. Flowavenet : A generative flow for raw audio, 2019.

[20] Kingma, D. P., and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions, 2018.

[21] Kingma, D. P., and Welling, M. Auto-encoding variational bayes, 2022.

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

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

[24] 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).

[25] Lu, W., Ng, H. T., Lee, W. S., and Zettlemoyer, L. S. A generative model for parsing natural language to meaning representations. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing* (Honolulu, Hawaii, Oct. 2008), Association for Computational Linguistics, pp. 783–792.

[26] Maspero, M., Savenije, M. H. F., Dinkla, A. M., Seevinck, P. R., Intven, M. P. W., Jurgenliemk-Schulz, I. M., Kerkmeijer, L. G. W., and van den

Berg, C. A. T. Dose evaluation of fast synthetic-ct generation using a generative adversarial network for general pelvis mr-only radiotherapy. *Physics in Medicine and Biology 63*, 18 (Sep 2018), 185001.

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

[28] Pan, Y., Liu, M., Lian, C., Zhou, T., Xia, Y., and Shen, D. Synthesizing missing pet from mri with cycle-consistent generative adversarial networks for alzheimer's disease diagnosis. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018* (Cham, 2018), A. F. Frangi, J. A. Schnabel, C. Davatzikos, C. Alberola-López, and G. Fichtinger, Eds., Springer International Publishing, pp. 455–463.

[29] Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation, 2018.

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

[31] Postels, J., Danelljan, M., Gool, L. V., and Tombari, F. Maniflow: Implicitly representing manifolds with normalizing flows, 2022.

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

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

[34] Seitzer, M. pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.3.0.

[35] Shukla, P., Pramanik, N., Mehta, D., and et al. Generative model based robotic grasp pose prediction with limited dataset. *Applied Intelligence 52* (07 2022), 9952–9966.

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

[37] Sun, H., Mehta, R., Zhou, H. H., Huang, Z., Johnson, S. C., Prabhakaran, V., and Singh, V. Dual-glow: Conditional flow-based generative model for modality transfer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2019).

[38] Tseng, B.-H., Cheng, J., Fang, Y., and Vandyke, D. A generative model for joint natural language understanding and generation, 2020.

[39] Wang, L., Chen, W., Yang, W., Bi, F., and Yu, F. R. A state-of-the-art review on image synthesis with generative adversarial networks. *IEEE Access 8* (2020), 63514–63537.

[40] Wang, Z., She, Q., and Ward, T. E. Generative adversarial networks in computer vision: A survey and taxonomy. *ACM Comput. Surv. 54*, 2 (feb 2021).

[41] Zhang, K., Ren, Y., Xu, C., and Zhao, Z. Wsrglow: A glow-based waveform generative model for audio super-resolution, 2021.

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