Stockholms universitet

# Bias-variance Tradeoff in Diffusion Models

Joakim Andersson Svendsen

Masteruppsats i matematisk statistik
Master Thesis in Mathematical Statistics

Masteruppsats 2026:1
Matematisk statistik
Februari 2026

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Matematiska institutionen

Stockholm University

Mathematical Statistics
Stockholm University
Master Thesis **2026:1**
http://www.math.su.se

# Bias-variance Tradeoff in Diffusion Models

Joakim Andersson Svendsen*

February 2026

**Abstract**

Denoising Diffusion Probabilistic Models (DDPMs) are state-of-the-art generative models whose training objective takes the form of a mean squared error. Despite this apparent similarity to a regression problem, the relevance of the classical bias–variance trade-off in DDPMs is not well understood. Unlike supervised learning, where prediction error and generalization can be directly assessed, generative models must be evaluated through indirect measures such as sample quality and diversity, complicating any direct transfer of bias-variance intuition.

This thesis examines how the bias-variance framework can be meaningfully interpreted in the context of DDPM training. We analyze the DDPM objective as a regression problem with a stochastic target and perform an empirical study on the MNIST dataset, focusing on training dynamics, noise-prediction residuals across diffusion timesteps, and the evolution of generated samples. Model behavior is assessed using loss-based diagnostics alongside established generative evaluation metrics, including the Inception Score and the Fr´echet Inception Distance.

The results indicate that persistent error in the learned denoising function is the dominant factor limiting sample fidelity, while variance-related effects such as training instability (e.g. mode collapse) are not observed in these experiments. Most improvements in generative quality occur early in training and then level off, with sample diversity preserved throughout. These findings suggest that the classical bias–variance trade-off does not carry over directly to diffusion models: bias primarily governs fidelity through persistent denoising error, whereas variance plays a secondary role by influencing training stability and, indirectly, diversity

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: Joakim.a.svendsen@gmail.com. Supervisor: Chun-Biu Li.

## Acknowledgements

I would like to sincerely thank my supervisor, Chun-Biu Li, for his guidance and support throughout this project. His supervision consistently focused on my learning and development, made himself available for discussion whenever needed, and provided thoughtful, constructive feedback that was invaluable throughout the thesis process. His supervision significantly contributed to my understanding of the subject and my overall growth.

I would also like to extend my gratitude to Nik Tavakolian for providing a trained MNIST classifier that was used in the validation experiments presented in this thesis, which shortened the evaluation process.

Finally, I note that while the conceptual content and original drafts of all figures are based on my own work and notes, AI-based tools were used to assist with coding and implementation when generating many of the figures.

# Contents

# 1 Introduction

Deep learning has emerged as a major area within modern machine learning, largely due to the strong empirical performance of neural networks on a broad range of tasks, including image recognition, speech processing, and natural language understanding [1]. Within this broader development, generative modeling has emerged as a particularly active area of research, driven by the goal of learning complex data distributions from which realistic new samples can be generated [1]. Among existing generative approaches, diffusion models have gained significant prominence and now represent the state of the art in image generation, owing to their strong empirical performance and training stability [2, 9]. Beyond images, diffusion-based models have also been successfully applied to domains such as audio and speech synthesis, video generation, and scientific applications including molecular design and protein structure modeling [8, 18].

Diffusion models generate data by reversing a gradual stochastic corruption process. Starting from a simple noise distribution, samples are transformed into structured data through a sequence of denoising steps [9]. Two closely related formulations dominate the literature: score-based diffusion models, which learn the score function of the data distribution via denoising score matching [18], and Denoising Diffusion Probabilistic Models (DDPMs), which define a discrete-time Markov chain and are trained to predict the noise added at each diffusion step [9]. This thesis focuses on DDPMs. A defining feature of DDPMs is that their training objective reduces to a mean squared error regression problem, where a neural network learns to predict a stochastic noise target conditioned on a noisy input. An equivalent regression interpretation also arises in score-based diffusion models under appropriate parameterizations, highlighting the close conceptual relationship between the two frameworks [9, 18].

In classical supervised learning, regression problems are commonly characterized by the bias–variance trade-off, which provides a principled framework for understanding underfitting, overfitting, and generalization [1]. Bias reflects systematic error arising from model limitations, while variance captures sensitivity to the particular training data. Extending this framework to generative models, however, is not straightforward. In generative settings, there is no ground-truth output (i.e. response variables) associated with a given input, and performance cannot be assessed through prediction error alone. Instead, model quality is evaluated using indirect criteria such as sample fidelity and diversity, making the relationship between bias, variance, and generalization more difficult to interpret [2, 7, 17].

The objective of this thesis is to investigate how the bias-variance framework can be meaningfully interpreted in the context of DDPMs. We combine a theoretical analysis of the DDPM training objective with an empirical study conducted on the MNIST dataset [12]. Training dynamics, noise-prediction residuals across diffusion timesteps, and the evolution of generated samples are examined alongside common generative evaluation metrics, including the Inception Score and the Fréchet Inception Distance [7, 17]. Through this analysis, we aim to clarify how bias and variance influence sample fidelity, diversity, and stability in diffusion models, and to identify the limitations of classical bias–variance interpretations in generative settings.

The remainder of this thesis is organized as follows. Chapter 2 introduces the theoretical foundations of neural networks, generative models, and diffusion models, and describes the network architecture, training configuration, and validation methods used in this work. Chapter 3 presents the empirical results and their analysis. Chapter 4 concludes with a discussion of the findings, their implications, and directions for future work.

# 2 Theory and Methods

### 2.0.1 Neurons and layers

Artificial neural networks form the basic computational framework used in modern deep learning. Their aim is to learn a parametric function $f_\theta$ that approximates an unknown target function $f$,

$$f_\theta(x) \approx f(x), \tag{1}$$

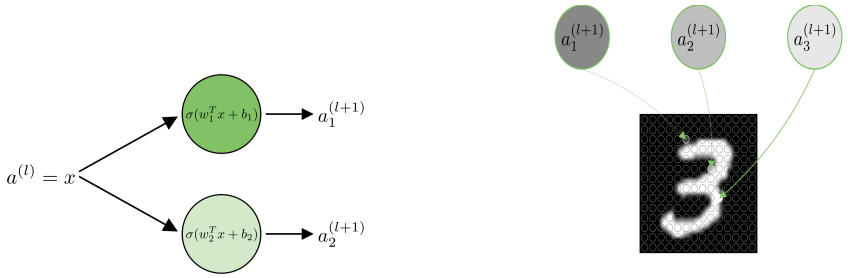
where $\theta$ denotes the collection of all trainable weights and biases. Training consists of adjusting these parameters so that the network captures salient structure present in the data, as measured by a chosen loss function and training objective.

At the most basic level, a neural network is built from neurons connected by weighted edges. The mapping itself is performed by these weighted connections together with a nonlinear activation function. Given an input vector $x \in \mathbb{R}^d$, a single neuron produces a scalar activation

$$a = \sigma(w^T x + b), \tag{2}$$

where $w \in \mathbb{R}^d$ denotes the weight vector associated with the incoming connections, $b \in \mathbb{R}$ is a bias term, and $\sigma(\cdot)$ is a nonlinear activation function. The inclusion of a nonlinearity prevents the network from collapsing into a purely linear model and allows it to represent nonlinear input–output relationships [5].

Neurons are arranged into layers, with multiple neurons in the same layer operating on a shared input. Figure 1a shows two neurons within a single layer receiving the same activation $a^{(l)} = x$. Each neuron applies its own weighted sum and activation function, producing different outputs. Although the input is identical, differences in the learned parameters lead to different responses: one neuron may respond strongly to a particular pattern, while another may respond only weakly. Taken together, these parallel responses allow a layer to encode several features of the input at once.




**(a)** **(b)**

**Figure 1:** Examples of neuron activations in a neural network. (a) Two neurons in the same layer responding differently to a shared input $\mathbf{a}^{(1)} = \mathbf{x}$. (b) Schematic illustration of neuron responses in the first layer to local pixel configurations in a handwritten digit. The illustration is conceptual and does not represent actual hidden-layer activations, which generally do not resemble the original input in pixel space.

Although the neuron model introduced above is simple, it already illustrates how neural networks extract structure from data. This can be made more concrete by considering an example from image data. Figure 1b shows a $14 \times 14$ grayscale image of a handwritten digit "3." Each pixel is treated as a numerical input, and the first layer processes these values without any explicit knowledge of image geometry. Instead, its neurons learn to respond to simple patterns in pixel intensities, such as local contrast changes or short stroke-like configurations. In this way, raw pixel values are transformed into internal representations through learned nonlinear mappings, even though these representations generally do not resemble the input image itself.

Neurons are organized into layers, forming a sequence of transformations. The input layer receives the raw data, which is then passed through one or more hidden layers. Each hidden layer applies a learned transformation that modifies and refines the representation. The final output layer produces task-dependent outputs, such as class probabilities in classification problems or real-valued predictions in regression.

Two basic structural properties are commonly used to describe a neural network:

- *Width*: the number of neurons within a layer, which affects how many distinct features can be represented at that stage.
- *Depth*: the total number of layers, which determines how many successive transformations the input undergoes.

In this thesis, we focus on feedforward neural networks, in which information flows strictly from input to output without feedback or recurrent connections, as this architecture forms the foundation for many modern deep learning models [1, 5].

For a feedforward network with $L$ layers, the overall mapping can be written as

$$f_\theta(x) = f^{(L)}\big(f^{(L-1)}(\cdots f^{(1)}(x)\cdots)\big), \tag{3}$$

where each $f^{(l)}$ denotes a layer-specific transformation parameterized by its own weights and biases. This layered composition allows the network to build complex mappings by combining simpler ones, with each layer producing a representation that serves as the input to the next.

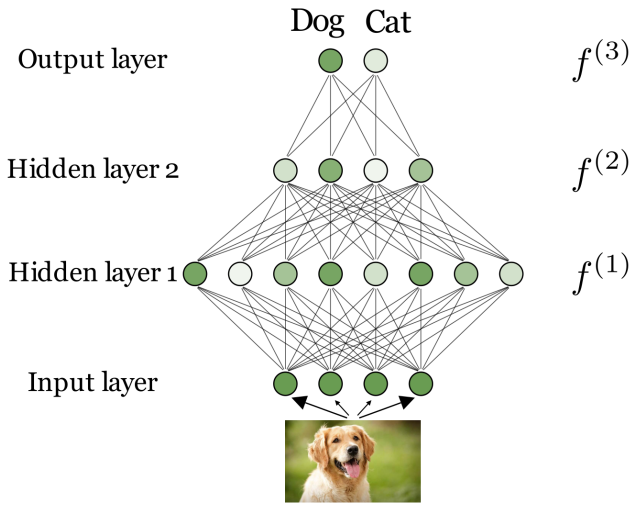



**Figure 2:** A feedforward neural network for image classification, illustrating the flow of information from the input image through multiple layers to class-specific outputs.

Figure 2 illustrates a feedforward neural network applied to an image classification task. An input image $x$ is processed sequentially by three layers according to

$$f_\theta(x) = f^{(3)}\big(f^{(2)}(f^{(1)}(x))\big), \tag{4}$$

so the network has depth three. The width is determined by the number of neurons in each layer. Individual neurons respond differently to various aspects of the input, and their activations are propagated forward through the network to produce the final class-specific outputs, here corresponding to the labels "Dog" and "Cat."

### 2.0.2 Loss function, backpropagation, and optimization

Training a neural network requires specifying an objective function that quantifies how well the model performs the task of interest. Learning is typically formulated as the optimization problem

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, f_\theta(x^{(i)})\big) + \lambda R(\theta), \tag{5}$$

where $f_\theta$ denotes a neural network parameterized by $\theta$, $\mathcal{L}(\cdot)$ is a loss function, and $R(\theta)$ is a regularization term that penalizes model complexity. The target $y^{(i)}$ depends on the learning setting. In supervised learning, it corresponds to a label, such as a class index or a real-valued response. In unsupervised, self-supervised, or generative settings, it may instead represent a reconstruction target, a similarity constraint, or an artificial signal such as injected noise, as in diffusion models.

Different learning objectives arise from different choices of the conditional output distribution. In all cases, the training loss is given by the cross-entropy (negative log-likelihood) between the data distribution and the model. For example, assuming Gaussian output noise in regression leads to squared or absolute error losses, respectively, while Bernoulli or categorical output distributions in classification give rise to binary or multiclass cross-entropy losses. Related likelihood-based objectives include reconstruction losses for autoencoders and noise-prediction losses for diffusion models. From the perspective of optimization, however, these objectives play a similar role: each defines a scalar function whose gradient with respect to the model parameters determines how the model is updated during training [1].

Gradients are computed efficiently using the backpropagation algorithm [5], which applies the chain rule to the layered structure of the network. Consider a feedforward neural network with layers indexed by $l = 1, \dots, L$, defined by

$$h^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}, \qquad a^{(l)} = \sigma\left(h^{(l)}\right), \tag{6}$$

where $a^{(0)} = x$ is the input, $h^{(l)}$ denotes the pre-activations, $a^{(l)}$ the activations, and $\sigma(\cdot)$ the activation function. Introducing the backpropagated gradient with respect to the pre-activations,

$$\delta^{(l)} := \frac{\partial \mathcal{L}}{\partial h^{(l)}}, \tag{7}$$

the gradients of the loss with respect to the parameters of layer $l$ are given by

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} \big(a^{(l-1)}\big)^\top, \qquad \frac{\partial \mathcal{L}}{\partial b^{(l)}} = \delta^{(l)}. \tag{8}$$

The backpropagated gradients propagate backward through the network according to

$$\delta^{(l)} = \big(W^{(l+1)}\big)^\top \delta^{(l+1)} \odot \sigma'\left(h^{(l)}\right), \tag{9}$$

where $\sigma'(\cdot)$ denotes the derivative of the activation function and $\odot$ indicates elementwise multiplication. The initialization at the output layer is determined by the choice of loss function and output parameterization.

Once gradients have been computed, the parameters are updated using gradient-based optimization. In its simplest form, gradient descent performs the update

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathcal{L}(\theta_k), \tag{10}$$

where $k \in \mathbb{N}$ indexes the optimization iteration, $\eta > 0$ is the learning rate and $\mathcal{L}(\theta)$ denotes the empirical risk, defined as the average loss over the full training dataset.

In practice, evaluating gradients over the entire dataset at each step is often impractical, and stochastic gradient descent (SGD) is used instead. Given a mini-batch $\mathcal{B}$, the update becomes

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \left( \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathcal{L}\big(y^{(i)}, f_\theta(x^{(i)})\big) \right). \tag{11}$$

Although SGD introduces randomness into the optimization process, it substantially reduces computational cost and often performs well in high-dimensional, non-convex settings. In many applications, it is combined with adaptive methods such as Adam, which incorporate momentum and parameter-specific learning rates [11].

Neural networks are highly expressive models and can overfit when their capacity is large relative to the amount of available data. Regularization is therefore used to improve generalization by discouraging overly complex solutions. One common approach is explicit parameter regularization through the penalty term $\lambda R(\theta)$. Typical choices include $R(\theta) = \|\theta\|_2^2$, which penalizes large parameter values, and $R(\theta) = \|\theta\|_1$, which promotes sparsity [1].

Regularization can also be introduced through the training procedure itself. Early stopping limits overfitting by terminating training before full convergence. Dropout randomly deactivates units during training, reducing reliance on specific activations. Data augmentation increases the effective size of the training set by applying label-preserving transformations to the inputs. While these techniques differ in implementation, they all act to constrain effective model complexity and stabilize training [5].

## 2.1 Generative models

A generative model seeks to learn a probability distribution over data, typically denoted by $p_\theta(x)$, or a conditional distribution $p_\theta(x \mid c)$ when auxiliary information $c$, such as class labels or text prompts, is available. Once trained, the model can generate new samples by drawing from this learned distribution, rather than producing predictions for fixed inputs.

In many settings, modeling $p_\theta(x)$ directly is difficult because high-dimensional data exhibit complex, structured variability that is not easily captured by a single distributional form. A common approach is therefore to introduce latent variables $z$ that represent hidden factors influencing the observations. This yields models of the form

$$p_\theta(x) = \int p_\theta(x|z)p(z)dz \tag{12}$$

where $p(z)$ is a simple prior distribution and $p_\theta(x|z)$is a conditional model that maps latent factors to observations. By integrating over $z$, the model can represent complex data distributions as mixtures over simpler conditional components, with the latent variable providing an intermediate representation that accounts for variability in the observed data [1, 10].

Generative models are often categorized according to whether they define an explicit probability density. Likelihood-based models specify a tractable or approximate likelihood and include autoregressive models, normalizing flows, variational autoencoders, diffusion models, and energy-based models. In contrast, implicit generative models, such as generative adversarial networks (GANs), do not provide an explicit likelihood and are trained using objectives defined purely through generated samples.

Model quality in generative settings is commonly discussed in terms of fidelity and diversity. Fidelity refers to how realistic individual samples appear and how well they reflect the structure of the data distribution, while diversity describes the extent to which the model captures variability across the dataset rather than producing a narrow range of similar outputs. These two aspects are inherently linked: high fidelity without diversity corresponds to memorization, whereas high diversity without sufficient structure leads to unrealistic samples. Although fidelity and diversity do not coincide exactly with the classical notions of bias and variance, they play an analogous role when assessing generalization in generative models, where direct prediction error is unavailable.

This thesis focuses on likelihood-based generative models, and in particular on diffusion models. While diffusion models differ from classical latent-variable models such as variational autoencoders in their architecture and sampling procedure, they share a probabilistic foundation rooted in latent variables and variational reasoning. As shown in later sections, the training objective of diffusion models can be cast as a regression problem, providing a concrete setting in which questions related to bias and variance can be examined in a generative context.

## 2.2 Encoder–Decoder and Latent Variable Models

In latent-variable models, a common formulation is the encoder–decoder framework, often discussed in the context of autoencoders. An autoencoder is a model that learns a representation of the data by mapping inputs through an encoder into a lower-dimensional latent variable $z$, commonly referred to as the bottleneck. A decoder then maps this latent representation back to the original data space, with the goal of reconstructing the input. Training is driven by a reconstruction loss, which encourages the latent representation to retain information that is most relevant for reproducing the data.

The role of the latent representation can be understood intuitively through a simple analogy. A shadow is not the object itself, but it preserves enough structure to convey its essential shape. In a similar way, the latent space provides a compressed and abstract representation of the input, which the decoder uses to reconstruct the original data. Figure 3 illustrates the basic structure of an autoencoder.

Standard autoencoders are deterministic: each input is mapped to a single latent vector. For the purposes of this thesis, it is more relevant to consider a probabilistic variant, namely the variational autoencoder (VAE). In a VAE, the latent variable is treated probabilistically, so that each data point is associated with a distribution over latent variables rather than a fixed vector.


**Figure 3:** Structure of a basic autoencoder consisting of an encoder that maps data to a low-dimensional latent representation and a decoder that reconstructs the input from this representation.

As shown in Figure 4, the overall encoder–decoder structure is retained, but the bottleneck is replaced by a parametric probability distribution.

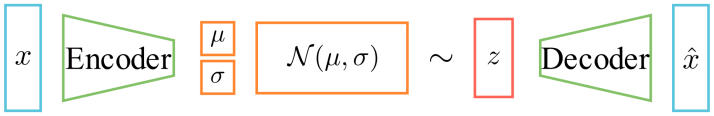



**Figure 4:** Structure of a variational autoencoder (VAE), where the deterministic bottleneck is replaced by a probabilistic latent distribution parameterized by a mean and variance.

Training a VAE is framed as a variational inference problem. The model specifies a prior distribution over latent variables, an approximate posterior, and a divergence term that measures the discrepancy between them. The role of this divergence is illustrated schematically in Figure 5, which shows how the Kullback–Leibler divergence quantifies the discrepancy between two probability distributions.

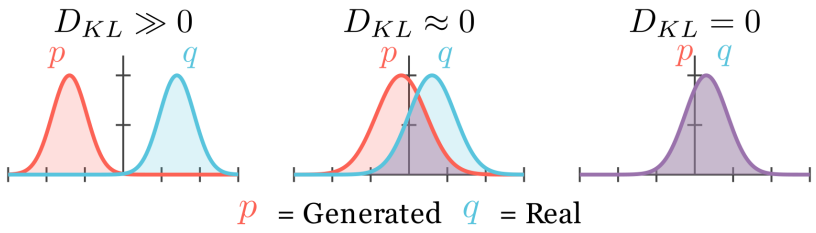



**Figure 5:** Illustration of the Kullback–Leibler divergence as a measure of discrepancy between probability distributions

To make this framework concrete, specific parametric forms must be chosen for both the prior and the approximate posterior. A common choice is

$$p(z) = \mathcal{N}(0, I), \qquad q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \sigma^2_\phi(x)), \tag{13}$$

where the mean and variance of the approximate posterior are parameterized by a neural network [10].

This choice is motivated by both modeling and computational considerations. The standard normal prior is simple and isotropic, while the Gaussian form of $q_\phi(z|x)$ allows the approximate posterior to flexibly adapt to the data through its learned mean and variance.

Under these assumptions, the Kullback–Leibler divergence $D_{\mathrm{KL}}(q_\phi(z \mid x)||p(z))$ admits a closed-form expression that depends only on $\mu_\phi(x)$ and $\sigma^2_\phi(x)$. As a result, this divergence can be evaluated exactly, avoiding the need for Monte Carlo estimation of this term and yielding a stable and efficient training objective, as originally proposed in the variational autoencoder framework [10].

As a likelihood-based generative model, a variational autoencoder aims to learn a probability distribution over the observed data [1, 10]. This corresponds to learning the marginal distribution

$$p_\theta(x) = \int p_\theta(x \mid z)p(z)\, dz, \tag{14}$$

which is generally intractable to compute directly. The difficulty arises because the integral requires marginalizing over all possible latent variable configurations $z$, and the conditional distribution $p_\phi(x|z)$ is parameterized by a nonlinear neural network. This combination prevents analytic evaluation of the integral and motivates the use of variational approximations and lower bounds, introduced in the following section.

Instead, variational autoencoders maximize a lower bound on the data log-likelihood,

$$\log p_\theta(x) \geq \mathcal{L}_{\mathrm{ELBO}}(\theta, \phi), \tag{15}$$

known as the Evidence Lower Bound (ELBO). This inequality follows from introducing an approximate posterior distribution $q_\phi(z|x)$ and applying Jensen's inequality to the marginal likelihood $\log p_\theta(x) = \log \int p_\theta(x|z)p(z)dz$ [10]. Equality holds if and only if the approximate posterior matches the true posterior, $q_\phi(z|x) = p_\theta(z|x)$ [1].

The ELBO takes the form

$$\mathcal{L}_{\mathrm{ELBO}}(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] - D_{\mathrm{KL}}(q_\phi(z \mid x)||p(z)). \tag{16}$$

and provides a tractable objective for learning[10]. The first term is the expected log-likelihood under the approximate posterior and encourages the decoder $p_\theta(x \mid z)$ to reconstruct the observed data from latent variables sampled according to $q_\phi(z \mid x)$. The second term is a Kullback–Leibler divergence that penalizes deviations of the approximate posterior from the prior $p(z)$, thereby regularizing the latent representation. Maximizing the ELBO therefore balances reconstruction accuracy against adherence to the chosen prior and jointly trains the decoder $p_\theta(x \mid z)$ and the encoder $q_\phi(z \mid x)$, requiring two neural networks.

Hierarchical variational autoencoders extend this framework by introducing multiple layers of latent variables rather than a single bottleneck. Diffusion models can be viewed as a special case of such hierarchical constructions, where each latent variable corresponds to a progressively noisier version of the data and the inference process is fixed rather than learned. This connection will be made precise in the following section.

## 2.3 Denoising Diffusion Probabilistic Models (DDPM)

So far, we have seen how generative models are formulated by introducing a latent variable distribution $p(z)$ and learning a mapping from this latent space to the data space $x$ using a neural network. While this approach has been highly successful, a different class of generative models has gained increasing attention in recent years: diffusion models. These models now represent the state of the art across a broad range of applications, including image generation, audio and speech synthesis, text-to-video generation, and scientific domains such as protein structure modeling and molecular design [9, 18].

The underlying idea of diffusion models is straightforward. Rather than generating data in a single step from a latent variable, diffusion models construct a sequence of latent variables by gradually corrupting the data. Starting from a data point $x_0$, noise is added over a sequence of steps, known as the forward diffusion process. At each step, the signal is both perturbed by noise and rescaled, so that information from the original data is progressively attenuated. After sufficiently many steps, the distribution approaches a simple Gaussian. An example of this progressive corruption is shown in Figure 6. The generative problem is then recast as learning the reverse of this process: a neural network is trained to remove noise step by step, transforming an initial Gaussian sample back into a realistic data point.

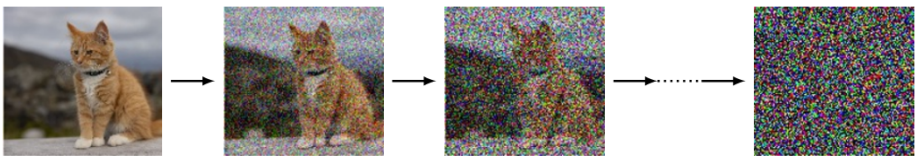


**Figure 6:** Example of progressive corruption of an image under the forward diffusion process, where Gaussian noise is added over successive timesteps. Adapted from Bishop and Bishop (2024), Chapter 20. [1].

Diffusion models are likelihood-based and are trained using a variational objective derived from this forward–reverse construction. A fixed and analytically tractable forward process defines how noise is added, while a neural network is trained to model the reverse transitions.

Unlike variational autoencoders, no learned inference network is required: the forward noising process is fully specified in advance, and all stochasticity in the latent variables is due to injected noise rather than learned uncertainty. This separation leads to a comparatively simple optimization problem and contributes to the stable training behavior observed in practice. Empirically, diffusion models have been shown to match or surpass the sample quality of generative adversarial networks while avoiding adversarial training instabilities [2].

Denoising Diffusion Probabilistic Models (DDPMs), which are the focus of this thesis, implement this idea in discrete time by defining a finite Markov chain

$$x_0 \to x_1 \to \cdots \to x_T, \tag{17}$$

where each transition adds a small amount of Gaussian noise [9]. The forward diffusion process is fixed and admits closed-form expressions, while the reverse process is learned from data. Throughout this chapter, distributions associated with the forward process and its conditionals are denoted by $q(\cdot)$, whereas the learned reverse generative model is parameterized by $p_\theta(\cdot)$. We begin by formalizing the forward diffusion process before introducing the parameterized reverse dynamics.

### 2.3.1 Forward process

We model the forward diffusion process as a Markov chain that progressively corrupts the data by adding Gaussian noise according to a predetermined variance schedule $\beta_1, \ldots, \beta_T$. This forward process is fixed and fully specified, and unlike the approximate posterior in variational autoencoders, it is not learned from data. Its role is instead to define a controlled noising procedure that yields analytically tractable transition densities. The forward transitions are defined as

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathcal{I}), \tag{18}$$

where $\mathcal{I}$ denotes the identity matrix. The mean term makes explicit that each step rescales the previous state before noise is added, ensuring that the signal magnitude is gradually attenuated as $t$ increases.

Starting from an initial data point $x_0 \in \mathbb{R}^d$, the noisy sequence $x_1, x_2, \ldots, x_T$ is generated recursively according to

$$x_t = \sqrt{1-\beta_t}, x_{t-1} + \sqrt{\beta_t}, \epsilon_t, \tag{19}$$

where each $\epsilon_t$ is an independent standard Gaussian noise variable [9]. Throughout this section, we assume $x_t \in \mathbb{R}^d$ and $\epsilon_t \sim \mathcal{N}(0, \mathcal{I}_d)$, with all Gaussian distributions being multivariate and isotropic.

Figure 7 illustrates the effect of this process: as $t$ increases, repeated rescaling and Gaussian perturbations progressively suppress the structure present in the original data. For sufficiently large $t$, the distribution of $x_t$ approaches a standard multivariate Gaussian, meaning that the sample contains no information about the original data point beyond random noise.

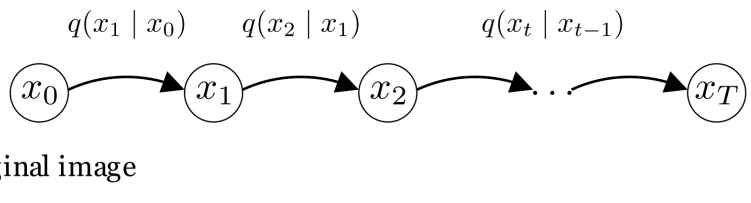



**Figure 7:** Forward diffusion (encoding) process from the original data point $x_0$ to a highly noisy latent variable $x_T$ through successive Gaussian perturbations.

Our goal is to express $x_t$ directly in terms of the original data point $x_0$, yielding a closed-form marginal distribution for the forward process. While this could be obtained by repeatedly applying Eq. (19), doing so quickly becomes cumbersome. To simplify the derivation, we introduce the notation

$$\alpha_t := 1 - \beta_t, \tag{20}$$

so that Eq. (19) can be rewritten as

$$x_t = \sqrt{\alpha_t}, x_{t-1} + \sqrt{1-\alpha_t}, \epsilon_t. \tag{21}$$

The first term scales down the contribution of the previous state, while the second term injects Gaussian noise. As $t$ increases, the contribution of the original signal is gradually reduced and the state becomes increasingly dominated by noise.

Using this form, we can express $x_t$ in terms of earlier states. For the previous timestep,

$$x_{t-1} = \sqrt{\alpha_{t-1}}, x_{t-2} + \sqrt{1-\alpha_{t-1}}, \epsilon_{t-1}. \tag{22}$$

Substituting into Eq. (21) gives

$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1-\alpha_t}\epsilon_t \tag{23}$$

$$= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-1} + \sqrt{1-\alpha_t}\epsilon_t. \tag{24}$$

Since $\epsilon_{t-1}$ and $\epsilon_t$ are independent standard Gaussian variables, any linear combination

$$a\epsilon_{t-1} + b\epsilon_t \sim \mathcal{N}(0, (a^2+b^2)\mathcal{I}). \tag{25}$$

Here, $a = \sqrt{\alpha_t(1-\alpha_{t-1})}$ and $b = \sqrt{1-\alpha_t}$, which yields

$$a^2 + b^2 = \alpha_t(1-\alpha_{t-1}) + (1-\alpha_t) \tag{26}$$

$$= 1 - \alpha_t \alpha_{t-1}. \tag{27}$$

We can therefore rewrite $x_t$ as

$$x_t = \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \epsilon_{t,t-2} \tag{28}$$

where $\epsilon_{t,t-2} \sim \mathcal{N}(0, \mathcal{I})$ denotes an aggregated Gaussian noise term.

Repeating this argument iteratively yields the same structure at arbitrary depth. It can be shown by induction that for any $k < t$,

$$x_t = \sqrt{\alpha_t \alpha_{t-1} \dots \alpha_{k+1}} x_k + \sqrt{1 - \alpha_t \alpha_{t-1} \dots \alpha_{k+1}} \epsilon_{t,k}, \tag{29}$$

with $\epsilon_{t,k} \sim \mathcal{N}(0, \mathcal{I})$. A detailed proof is given in Higham et al. (2023, Section 3) [8].

Finally, defining

$$\bar{\alpha}_t := \prod_{i=1}^{t} \alpha_i, \tag{30}$$

we obtain the closed-form expression

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \bar{\epsilon}_t, \tag{31}$$

where $\bar{\epsilon}_t \sim \mathcal{N}(0, \mathcal{I})$. Equivalently, the marginal forward transition can be written as

$$q(x_t | x_0) := \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)\mathcal{I}). \tag{32}$$

This closed-form characterization of the forward process allows Gaussian noise to be added to the data at any timestep directly, without simulating all intermediate steps. This property is central to the efficiency and tractability of diffusion models and will be used extensively in the construction of the training objective.

### 2.3.2 Reverse process

In the reverse process, the objective is to characterize the distribution of the previous state $x_{t-1}$ given a noisy observation $x_t$. For the fixed forward diffusion process, this reverse-time transition can be expressed in closed form only when conditioning additionally on the original data point $x_0$. We therefore consider the conditional distribution

$$q(x_{t-1} | x_t, x_0), \tag{33}$$

which represents the true reverse-time transition implied by the forward diffusion process [9]. Conditioning on $x_0$ reflects the fact that, under the forward process, $x_t$ retains information about the original data point, and the exact posterior over earlier states is therefore defined with respect to $x_0$.

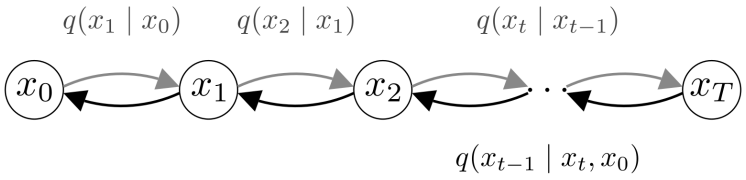



**Figure 8:** Overview of the diffusion framework, showing the forward diffusion (encoding) process that adds noise and the reverse diffusion (decoding) process that progressively removes noise to generate data

Figure 8 provides an overview of this framework. Although the reverse direction in the figure corresponds to the denoising procedure used for generation, we begin by deriving the exact reverse conditional associated with the forward process. This conditional assumes access to the clean data point $x_0$ and serves as a reference distribution that motivates the learned reverse process introduced later.

Using Bayes' rule together with the Markov structure of the forward process, the reverse conditional can be written as

$$q(x_{t-1} \mid x_t, x_0) = \frac{q(x_t \mid x_{t-1}, x_0) q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)} \quad \text{(Bayes)} \tag{34}$$

$$= \frac{q(x_t \mid x_{t-1}) q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)} \quad \text{(Markov)}. \tag{35}$$

where the second equality follows from the Markov property of the forward diffusion chain.

Substituting the Gaussian transition densities derived earlier yields

$$q(\cdot) = \frac{\mathcal{N}\left(x_t; \sqrt{\alpha_t}, x_{t-1}, (1-\alpha_t)\mathcal{I}\right) \mathcal{N}\left(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}, x_0, (1-\bar{\alpha}_{t-1})\mathcal{I}\right)}{\mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}, x_0, (1-\bar{\alpha}_t)\mathcal{I}\right)} \tag{36}$$

$$\propto \mathcal{N}\left(x_t; \sqrt{\alpha_t}, x_{t-1}, (1-\alpha_t)\mathcal{I}\right) \mathcal{N}\left(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}, x_0, (1-\bar{\alpha}_{t-1})\mathcal{I}\right). \tag{37}$$

The denominator does not depend on $x_{t-1}$ and therefore acts only as a normalizing constant. Up to normalization, the reverse conditional is proportional to the product of two Gaussian densities in $x_{t-1}$.

The product of two Gaussian densities is itself proportional to a Gaussian. Writing the factors abstractly as $\mathcal{N}(x; \mu_1, \sigma_1^2)$ and $\mathcal{N}(x; \mu_2, \sigma_2^2)$, their product yields

$$\mathcal{N}(\mu_q, \sigma_q^2) = \mathcal{N}(\mu_1, \sigma_1^2)\mathcal{N}(\mu_2, \sigma_2^2) = \mathcal{N}\left(\frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_2^2 + \sigma_1^2}, \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\right). \tag{38}$$

Applying this result to the present setting gives the closed-form expressions

$$\mu_q(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t}, \tag{39}$$

$$\sigma_q^2(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}, \tag{40}$$

as derived in Appendix 5.2.

The exact reverse conditional distribution is therefore

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \mu_q(x_t, x_0), \sigma_q^2(t)\mathcal{I}), \tag{41}$$

and sampling from this distribution takes the form

$$x_{t-1} = \mu_q(x_t, x_0) + \sigma_q(t)z \qquad z \sim \mathcal{N}(0, \mathcal{I}). \tag{42}$$

Although this expression is exact, it is not directly usable during generation, since the clean data point $x_0$ is unknown at sampling time. To remove the explicit dependence on $x_0$, we use the forward-process identity

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t, \tag{43}$$

where $\bar{\epsilon}_t \sim \mathcal{N}(0, \mathcal{I})$ denotes the noise realization in the forward process. Since $x_0$ is not available at generation time, one replaces $x_0$ using the forward-process identity

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t^* \Longrightarrow x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_t^*). \tag{44}$$

where $\epsilon_t^*$ is a Gaussian noise variable with the same distribution as the forward-process noise.

Substituting this expression into $\mu_q(x_t, x_0)$ yields

$$\mu_q = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1})}(1-\alpha_t)x_0}{1-\bar{\alpha}_{t-1})} \tag{45}$$

$$= \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t^*\right). \tag{46}$$

The reverse sampling step can therefore be written as

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t^*\right) + \sigma_q(t)z \qquad z \sim \mathcal{N}(0, \mathcal{I}). \tag{47}$$

This form makes explicit that reverse diffusion consists of two components: a denoising step that subtracts an estimate of the noise present in $x_t$, followed by the addition of appropriately scaled Gaussian noise [9]. In the next section, this observation will motivate the parameterization of the learned reverse process in terms of a noise-prediction network.

### 2.3.3 ELBO and loss function

In the previous section, we derived the exact reverse conditional implied by the forward diffusion process,

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\left(x_{t-1};\, \mu_q(x_t, x_0), \sigma_q^2(t)\mathcal{I}\right), \tag{48}$$

where

$$\mu_q(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_t^*\right), \tag{49}$$

$$\sigma_q^2(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}, \tag{50}$$

and

$$\epsilon_t^* = \frac{x_t - \sqrt{\bar{\alpha}_t}\,x_0}{\sqrt{1-\bar{\alpha}_t}}. \tag{51}$$

This describes the optimal reverse step when the clean point $x_0$ is known. During generation, however, $x_0$ is unavailable, so neither $\epsilon_t^*$ nor $\mu_q(x_t, x_0)$ can be evaluated. The practical goal is therefore to build a parameterized reverse process with the same Gaussian form, but depending only on what is observed at sampling time, namely $(x_t, t)$. Since the reverse variance $\sigma_q^2(t)$ is determined entirely by the forward schedule, learning is concentrated in the reverse mean.

DDPMs implement this idea by introducing a neural network $\epsilon_\theta(x_t, t)$ and defining a learned reverse kernel $p_\theta(x_{t-1} \mid x_t)$ [9]. The forward process is fixed, so only this single network is trained, in contrast to variational autoencoders which require separate encoder and decoder networks.

Replacing the unknown noise $\epsilon_t^*$ by the network prediction $\epsilon_\theta(x_t, t)$ gives

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1};\, \mu_\theta(x_t, t), \sigma_t^2\mathcal{I}\right), \tag{52}$$

with

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_\theta(x_t, t)\right). \tag{53}$$

Thus, the denoising dynamics are governed by the network through the mean, while the variance $\sigma_q(t)^2$ is fixed and determined by the forward diffusion process. In particular, for DDPMs one commonly chooses $\sigma_q(t)^2 = \beta_t$, which corresponds to the variance of the forward transition at timestep $t$ [9].

As a likelihood-based generative model, a DDPM is trained by maximizing the data log-likelihood $\log p_\theta(x_0)$. The model defines a marginal distribution over data by integrating out the latent diffusion variables,

$$p_\theta(x_0) = \int p_\theta(x_{0:T})\,\mathrm{d}x_{1:T}, \tag{54}$$

where the joint distribution factorizes as

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t). \tag{55}$$

Computing $\log p_\theta(x_0)$ exactly is intractable because it requires integrating over all diffusion trajectories $x_{1:T}$ that could lead to the same observation $x_0$.

$$-\log p_\theta(x_0) = -\log \int p_\theta(x_{0:T})\,\mathrm{d}x_{1:T}. \tag{56}$$

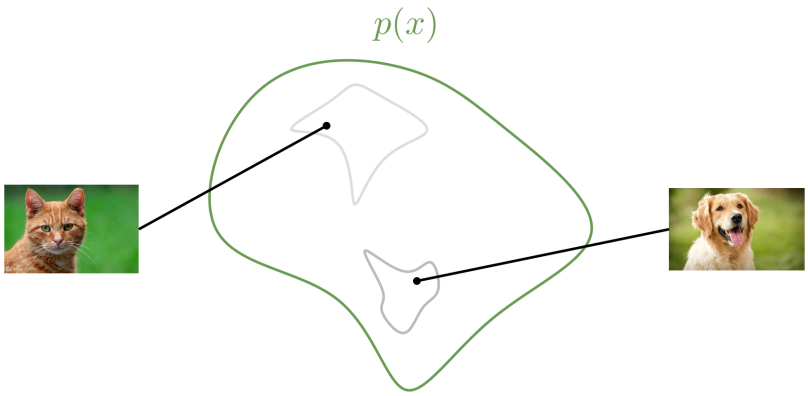



**Figure 9:** Schematic illustration of the target data distribution $p(x)$: individual observations (e.g., cat and dog images) lie on a lower-dimensional manifold within the ambient input space.

Figure 9 sketches the geometric intuition underlying this intractability. Real data are concentrated on a small, structured subset of the ambient space, whereas the prior distribution $p(x_T)$ used for generation is a simple, typically isotropic Gaussian that assigns mass broadly across the space. During generation, the model must transform samples from this diffuse prior into samples lying on the data manifold, so that many distinct stochastic paths can collapse into the same high-density data region. Because the diffusion process is stochastic, there are many possible latent trajectories $x_{1:T}$ that can connect a given noise sample $x_T$ to the same data point $x_0$, as Figure 10 shows. As the number of diffusion steps $t$ increases, the number of such trajectories grows rapidly, making exact marginalization over all possible paths computationally infeasible.

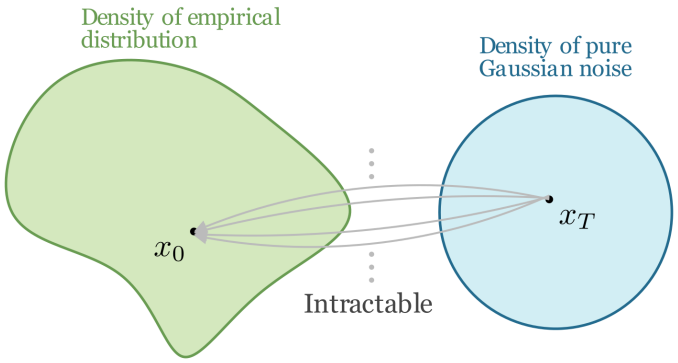



**Figure 10:** Latent diffusion paths: schematic of the many possible trajectories from the prior $p(x_T)$ to a data point $x_0$, motivating variational training via an ELBO.

This motivates replacing direct likelihood maximization with a variational objective that provides a tractable bound on the data log-likelihood. Starting from

$$
\begin{aligned}
-\log p_\theta(x_0) &= -\log \int p_\theta(x_{0:T})\,\mathrm{d}x_{1:T} && (57)\\
&= -\log \int q(x_{1:T}\mid x_0)\,\frac{p_\theta(x_{0:T})}{q(x_{1:T}\mid x_0)}\,\mathrm{d}x_{1:T} && (58)\\
&= -\log \mathbb{E}_{q(x_{1:T}|x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T}\mid x_0)}\right] && (59)\\
&\le -\mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}\mid x_0)}\right], \qquad \text{(Jensen's inequality)} && (60)
\end{aligned}
$$

we obtain an upper bound on the negative log-likelihood. Equivalently, the negative of the right-hand side is a lower bound on $\log p_\theta(x_0)$, commonly called the Evidence Lower Bound (ELBO).

After expanding and rearranging the ELBO (see Appendix 5.2), the bound can be written as

$$
\begin{aligned}
&\mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}\mid x_0)}\right] = \\
&= \mathbb{E}_{q(x_{1:T}|x_0)}\Bigg[D_{KL}\left(q(x_T\mid x_0)\,||\,p(x_T)\right) && (61)\\
&+ \sum_{t>1} D_{KL}\left(q(x_{t-1}\mid x_t, x_0)\,||\,p_\theta(x_{t-1}\mid x_t)\right) && (62)\\
&- \log p_\theta(x_0\mid x_1)\Bigg]. && (63)
\end{aligned}
$$

The first term depends only on the fixed forward process and the prior, and is therefore independent of $\theta$. The final term corresponds to the likelihood contribution at the final denoising step and typically has limited influence on learning. The main training signal comes from the sum of KL divergences that match the learned reverse transitions to the true reverse conditionals.

When both conditionals are Gaussian, the KL divergence has a closed form. If we fix the reverse variance so that $\Sigma_\theta = \sigma_t^2 \mathcal{I}$, then constant terms cancel and

$$\begin{aligned} &D_{\mathrm{KL}}\left(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)\right) = \\ &= D_{\mathrm{KL}}\left(\mathcal{N}(\mu_q, \sigma_q^2(t)\mathcal{I}) \,||\, \mathcal{N}(\mu_\theta, \sigma_t^2 \mathcal{I})\right) && (64) \\ &= \frac{1}{2}\left(\mu_\theta - \mu_q\right)^\top \Sigma_\theta^{-1} (\mu_\theta - \mu_q) && (65) \\ &= \frac{1}{2\sigma_t^2} ||\mu_q - \mu_\theta||^2, && (66) \end{aligned}$$

Consequently, the ELBO training objective reduces to minimizing a weighted squared error between the true and learned reverse means, averaged over the forward diffusion process,

$$\mathbb{E}_{q(x_{0:T})}\left[\sum_{t>1} \frac{1}{2\sigma_t^2} ||\mu_q(x_t, x_0) - \mu_\theta(x_t, t)||^2\right]. \tag{67}$$

Here, the expectation is taken with respect to the joint distribution $q(x_{0:T})$ induced by the fixed forward process [9].

Substituting the expression for $\mu_q(x_t, x_0)$ yields a loss that can be written directly in terms of noise prediction,

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{q(x_{0:T})}\left[\sum_{t>1} \frac{1}{2\sigma_t^2} ||\mu_q(x_t, x_0) - \mu_\theta(x_t, t)||^2\right] && (68) \\ &= \mathbb{E}_{q(x_{0:T})}\left[\sum_{t>1} \frac{(1-\alpha_t)^2}{2\sigma_t^2\, \alpha_t(1-\bar{\alpha}_t)} ||\epsilon - \epsilon_\theta(x_t, t)||^2\right]. && (69) \end{aligned}$$

Computing the full sum over all timesteps for every training example is computationally expensive. In practice, one samples a single timestep $t$ (typically uniformly from $\{1, \ldots, T\}$ for each data point and each iteration. Averaged over training iterations, this stochastic objective corresponds to the full sum above, yielding

$$L(\theta) := \mathbb{E}_{q(x_0),t,\epsilon}\left[w_t ||\epsilon - \epsilon_\theta(x_t, t)||^2\right], \qquad w_t := \frac{(1-\alpha_t)^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)}. \tag{70}$$

Equivalently, the objective can be written as an average over per-timestep losses,

$$L(\theta) = \mathbb{E}_t[\mathcal{L}_t(\theta)], \qquad \mathcal{L}_t(\theta) := \mathbb{E}_{q(x_0),\epsilon}[w_t||\epsilon - \epsilon(x_t, t)||^2], \tag{71}$$

where $\mathcal{L}_t(\theta)$ denotes the loss associated with denoising at a fixed timestep $t$.

For small $t$, the factor $1 - \bar{\alpha}_t = 1 - \prod_{i=1}^{t}(1 - \beta_i)$ is small, which can make the weight $w_t$ large. This places substantially more emphasis on low-noise denoising steps than on high-noise steps. Ho et al. report that dropping the timestep-dependent weights $w_t$ and training with an unweighted mean squared error often leads to better empirical sample quality [9]. From an optimization perspective, this simplification is justified because each $\mathcal{L}_t(\theta)$ is minimized independently: since the network explicitly conditions on $t$, the minimizer of $w_t\mathcal{L}_t(\theta)$ is identical to the minimizer of $\mathcal{L}_t(\theta)$ for each fixed $t$. The weights $w_t$ therefore only affect the relative scaling of gradients across timesteps, not the location of the optimum.

This motivates the commonly used simplified objective

$$\mathcal{L}_{\text{simple}}(\theta) = \mathbb{E}_{q(x_0),t,\epsilon}\left[||\epsilon - \epsilon_\theta(x_t, t)||^2\right], \tag{72}$$

which retains the same minimizer as the weighted objective up to a timestep-dependent scaling, but yields more stable optimization in practice.

This shows that DDPM training can be viewed as a regression problem: the network is trained to predict the forward-process noise from the corrupted input $(x_t, t)$ by minimizing a mean squared error. Through the noise prediction $\epsilon_\theta(x_t, t)$ the model implicitly specifies the reverse-process mean $\mu_\theta(x_t, t)$, and thereby the denoising transitions that define the generative process. Sampling is performed by iteratively applying the learned reverse transitions, starting from a noise sample $x_T \sim \mathcal{N}(0, \mathcal{I})$ and proceeding backwards in time until $x_0$ is obtained. Figure 11 summarizes the full framework, including the forward noising process used during training, the learned reverse denoising transitions, and the resulting sampling procedure.

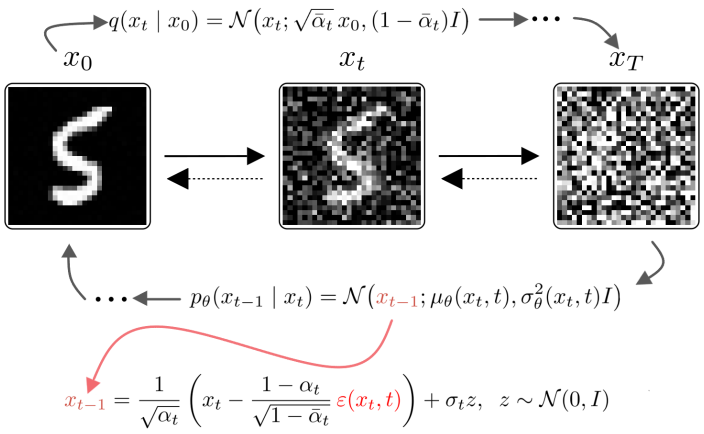



**Figure 11:** Overview of the Denoising Diffusion Probabilistic Model (DDPM). The forward process progressively corrupts a data sample $x_0$ through Gaussian transitions $q(x_t|x_{t-1})$; the marginal distribution $q(x_t|x_0)$ is shown for a representative intermediate timestep $t$, with the remaining forward steps proceeding up to the terminal noise sample $x_T$. The reverse process is defined by learned Gaussian transitions $p_\theta(x_{t-1}|x_t)$, whose mean is parameterized via the noise prediction network $\epsilon_\theta(x_t, t)$. Sampling corresponds to iteratively applying these reverse transitions, starting from an initial noise sample $x_T$ and proceeding backwards in time until a sample $x_0$ is obtained.

## 2.4 Network Architecture

### 2.4.1 Convolutional Networks

In Section 2.0.1, we considered fully connected neural networks for image recognition tasks, such as identifying a dog in an image. While the idea is simple, fully connected architectures are rarely used in practice for image data. Images are high-dimensional and structured, and a fully connected layer links every pixel to every hidden unit. This leads to an impractically large number of parameters, making training expensive and increasing the risk of overfitting [5]. More importantly, treating an image as an unordered vector ignores the spatial layout of pixels and fails to take advantage of locality and hierarchical structure.

An image is not an arbitrary point in $\mathbb{R}^d$. It is a two-dimensional grid (or three-dimensional when channels are included) in which nearby pixels are typically correlated. Many visual patterns are local: edges, corners, and textures are detected in small neighborhoods and subsequently combined to form larger structures such as shapes and objects. Ignoring this spatial organization forces a model to relearn the same local patterns independently at many locations, leading to inefficient use of both data and parameters.

Convolutional neural networks (CNNs) address this structure by introducing architectural constraints that reflect the local and repetitive nature of visual features. In particular, convolutional layers employ sparse connectivity, restricting each unit to depend only on a local receptive field, and parameter sharing, whereby the same set of weights is applied across all spatial locations. Together, these design choices enable convolutional layers to represent visual patterns efficiently while drastically reducing the number of free parameters [5].

The central operation is a convolution-like mapping that applies a small filter, or kernel, across the image to produce a feature map. Intuitively, the kernel acts as a pattern detector that is reused at every spatial location. Formally, for an input image $I$ with pixel values $I(j,k)$ and a kernel $K$ with entries $K(l,m)$, the feature map $C$ is defined by

$$C(j,k) = \sum_{l} \sum_{m} I(j+l,\, k+m)\, K(l,m). \tag{73}$$

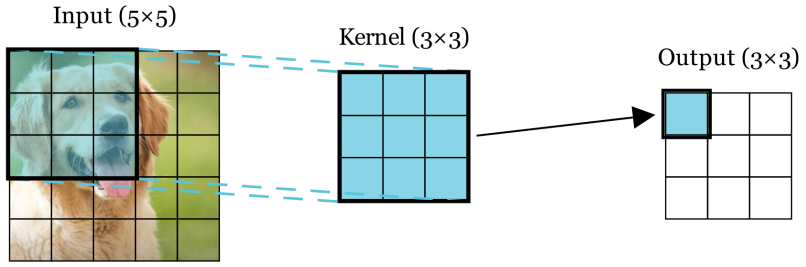



**Figure 12:** Discrete convolution operation. A local input patch is multiplied elementwise with a convolutional kernel and summed to produce a single feature-map activation.

Figure 12 shows how a local patch is combined with the kernel to produce one activation in the feature map, and Figure 13 gives a numerical example of the same computation.

The operation in Eq. (73) is the basic building block of convolutional neural networks. Its most important property is translational equivariance: the same kernel is applied at every spatial location, so shifting the input image leads to a corresponding shift in the feature map. This behavior is a direct consequence of parameter sharing, since the kernel weights are identical at all positions. This means that convolutional layers can detect the same local pattern regardless of where it appears in the image, without requiring separate parameters for each possible location [5].

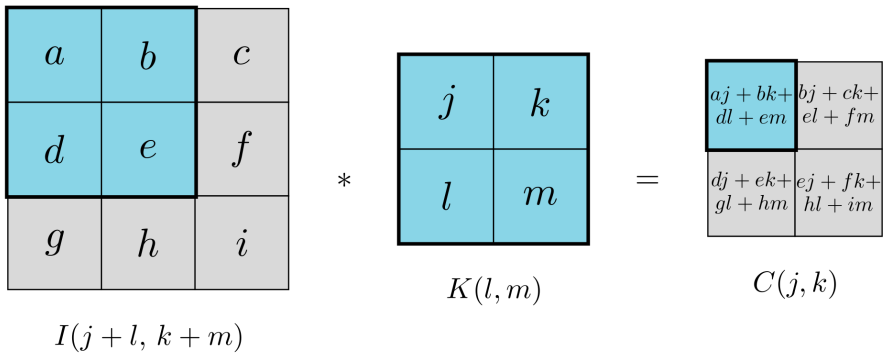



**Figure 13:** Numerical example of a convolution operation, showing how kernel weights and a local input patch combine to produce an output value.

Stacking convolutional layers also supports a natural feature hierarchy. Early layers tend to respond to simple local structures such as edges, while deeper layers combine these responses into more complex patterns, such as shapes or object parts. This layered composition is one of the reasons CNNs work well on images.

Two hyperparameters that strongly affect a convolutional layer are stride and padding. The stride determines how far the kernel is shifted between evaluations. With stride $s = 1$, the kernel moves one pixel at a time, producing overlapping receptive fields and a dense output map, as shown in Figure 14.

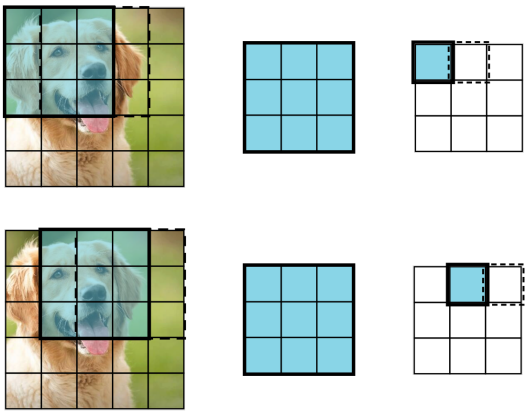


**Figure 14:** Effect of stride parameter $s = 1$ in convolution. The filter is shifted one pixel at a time, producing overlapping receptive fields and dense output activations.

Padding addresses what happens at the image boundaries. Without padding, the kernel cannot be centered near the border, so the output feature map shrinks relative to the input. A common fix is zero-padding, where extra pixels are added around the image before applying the kernel. With suitable padding, the output resolution can be preserved and border regions are treated more consistently. Figure 15 illustrates this idea.

The region of the input that influences a particular activation is called its receptive field. As convolutional layers are stacked, receptive fields grow: an activation in a deeper layer depends on a larger part of the original image. Figure 16 illustrates this growth for three layers using kernel size $k = 3$ and stride $s = 1$.

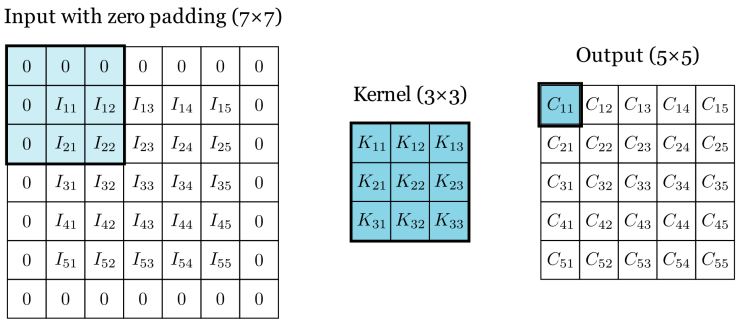



**Figure 15:** Zero padding in convolution. Padding extends the input at the image borders to counteract the reduction in spatial dimensions caused by convolution, helping to preserve the spatial size of feature maps across successive layers.

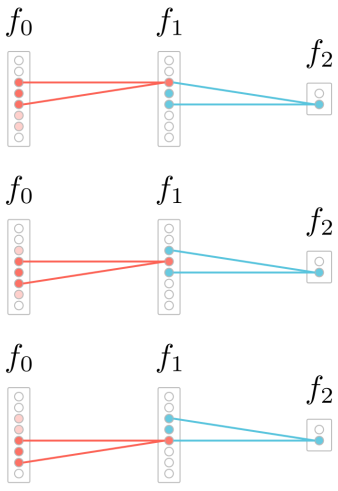



**Figure 16:** Growth of receptive fields across convolutional layers. Deeper-layer activations depend on increasingly larger regions of the original input.

More generally, for kernel sizes $k_l$ and strides $s_l$, the receptive field size after $L$ layers can be computed as

$$r_0 = \sum_{l=1}^{L} \left( (k_l - 1) \prod_{i=1}^{l-1} s_i \right) + 1. \tag{74}$$

Here $r_0$ denotes the size of the receptive field measured in the input layer (layer 0), i.e., the number of input pixels that influence a single activation at the last layer $L$.

As a simple example, consider repeated convolutions with constant kernel size $k_l = 3$ and stride $s_l = 1$ applied to a $5 \times 5$ image (like the dog image in Figure 12). In this case, (74) reduces to

$$r_0 = \sum_{l=1}^{L} (3 - 1) \cdot 1 + 1 = 2L + 1. \tag{75}$$

To cover an entire $5 \times 5$ input (i.e., $r_0 = 5$), one needs $2L + 1 = 5$, which gives $L = 2$ layers.

While informative, this toy example does not reflect real images, which often contain hundreds or thousands of pixels per dimension. In practice, relying only on depth to expand receptive fields would make networks unnecessarily large and computationally inefficient.

To capture larger context while keeping computation manageable, CNNs have other alternatives in downsampling to reduce the spatial resolution of feature maps. Downsampling by a factor $s > 1$ maps an input of size $H \times W$ to

$$H \times W \longrightarrow \frac{H}{s} \times \frac{W}{s}. \tag{76}$$

Common approaches are pooling and strided convolutions.

Pooling performs fixed downsampling by applying a predefined aggregation over local neighborhoods, typically a maximum or an average. This reduces spatial resolution without adding learnable parameters. Figure 17 shows examples of max pooling and average pooling.

Strided convolutions reduce spatial resolution by evaluating the convolution kernel at spaced locations while still performing feature extraction. For stride $s = 2$, the operation takes the form

$$C(j, k) = \sum_{l} \sum_{m} I(2j + l,\, 2k + m)\, K(l, m). \tag{77}$$

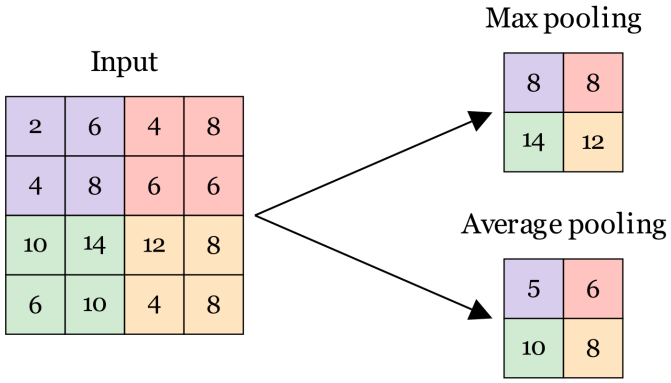



**Figure 17:** Pooling operations in convolutional networks. Max pooling selects the maximum value and average pooling computes the mean within local neighborhoods to reduce spatial resolution.

Instead of sliding the filter across every pixel, the kernel is shifted two pixels at a time in each spatial direction. This effectively downsamples the feature map by a factor of two while allowing the network to learn which local patterns should be retained as resolution decreases. Figure 18 illustrates this case.

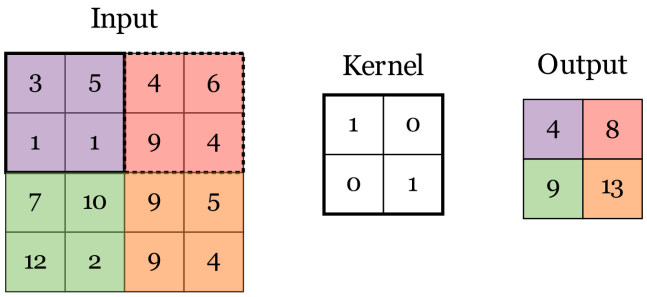



**Figure 18:** Strided convolution with stride $s = 2$. The filter is applied at spaced locations, reducing spatial resolution while performing feature extraction.

To make the effect of striding concrete, we consider the numerical example shown in Figure 18. The input image and kernel are given by

$$I = \begin{bmatrix} I(0,0) & I(0,1) & I(0,2) & I(0,3) \\ I(1,0) & I(1,1) & I(1,2) & I(1,3) \\ I(2,0) & I(2,1) & I(2,2) & I(2,3) \\ I(3,0) & I(3,1) & I(3,2) & I(3,3) \end{bmatrix} = \begin{bmatrix} 3 & 5 & 4 & 6 \\ 1 & 1 & 9 & 4 \\ 7 & 10 & 9 & 5 \\ 12 & 2 & 9 & 4 \end{bmatrix}, \tag{78}$$

$$K = \begin{bmatrix} K(0,0) & K(0,1) \\ K(1,0) & K(1,1) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{79}$$

With stride $s = 2$, the resulting feature map has spatial dimensions $2 \times 2$,

$$C = \begin{bmatrix} C(0,0) & C(0,1) \\ C(1,0) & C(1,1) \end{bmatrix} \tag{80}$$

since the kernel is evaluated only at the positions $(0,0), (0,2), (2,0), (2,2)$ in the input.

Each output entry is computed as

$$\begin{aligned} C(j,k) &= \sum_{l=0}^{1} \sum_{m=0}^{1} I(2j+l,\, 2k+m)\, K(l,m) && (81) \\ &= I(2j,2k)K(0,0) + I(2j,2k+1)K(0,1) \\ &\quad + I(2j+1,2k)K(1,0) + I(2j+1,2k+1)K(1,1), && (82) \end{aligned}$$

for $j, k \in \{0,1\}$. We then compute each output value

$$\begin{aligned} C(0,0) &= I(2\cdot 0+0, 2\cdot 0+0)K(0,0) + I(2\cdot 0+0, 2\cdot 0+1)K(0,1) && (83) \\ &\quad + I(2\cdot 0+1, 2\cdot 0+0)K(1,0) + I(2\cdot 0+1, 2\cdot 0+1)K(1,1) && (84) \\ &= I(0,0)K(0,0) + I(0,1)K(0,1) + I(1,0)K(1,0) + I(1,1)K(1,1) && (85) \\ &= 3\cdot 1 + 5\cdot 0 + 1\cdot 0 + 1\cdot 1 \\ &= 4. \end{aligned}$$

For the rest

$$\begin{aligned} C(0,1) &= I(0,2)K(0,0) + I(0,3)K(0,1) + I(1,2)K(1,0) + I(1,3)K(1,1) && (86) \\ &= 4\cdot 1 + 6\cdot 0 + 9\cdot 0 + 4\cdot 1 \\ &= 8, \end{aligned}$$

$$\begin{aligned} C(1,0) &= I(2,0)K(0,0) + I(2,1)K(0,1) + I(3,0)K(1,0) + I(3,1)K(1,1) && (87) \\ &= 7\cdot 1 + 10\cdot 0 + 12\cdot 0 + 2\cdot 1 \\ &= 9, \end{aligned}$$

$$\begin{aligned} C(1,1) &= I(2,2)K(0,0) + I(2,3)K(0,1) + I(3,2)K(1,0) + I(3,3)K(1,1) && (88) \\ &= 9\cdot 1 + 5\cdot 0 + 9\cdot 0 + 4\cdot 1 \\ &= 13. \end{aligned}$$

The final output feature map is therefore

$$\begin{bmatrix} 4 & 8 \\ 9 & 13 \end{bmatrix}. \tag{89}$$

This example shows how strided convolutions combine spatial downsampling with learned feature extraction. Rather than discarding information through fixed aggregation, as in pooling, the kernel weights determine which local patterns are retained as resolution is reduced. This ability to jointly perform downsampling and representation learning is central to the efficiency of modern convolutional architectures.

After repeated downsampling, it is often necessary to recover spatial resolution, especially in tasks where predictions are required at the pixel level. This is typically done using transposed convolutions, sometimes referred to as up-convolutions. While strided convolutions reduce resolution by skipping input locations, transposed convolutions perform the reverse operation by expanding a low-resolution feature map into a higher-resolution one using learned filters. Rather than relying on fixed interpolation, this allows the network to learn how coarse features should be mapped back to finer spatial grids. Transposed convolutions are therefore commonly used in decoder architectures to undo the effects of downsampling and are a standard component in models for dense prediction tasks such as semantic segmentation and image generation [21].

CNNs are widely used in image tasks because they are parameter-efficient and are built around assumptions that match the structure of image data. Architectures such as ResNet [6] and U-Net [15] build on these ideas through additional design choices, which are discussed in the next subsection.

### 2.4.2 U-Net for noise prediction

As discussed in Section 2.3.3, training a DDPM amounts to learning a function that predicts the noise realization $\epsilon(x_t, t)$ added at each diffusion step. This task requires a neural network that can identify noise patterns across multiple spatial scales while being explicitly conditioned on the diffusion timestep. In practice, this fitting task is quite often modelled by a U-Net architecture. Its multi-resolution structure aligns well with the progressive denoising nature of diffusion models, and its fully convolutional design allows predictions to be made at the original image resolution without introducing resolution-dependent fully connected layers.

The U-Net was originally proposed for image segmentation, but it can more generally be viewed as a convolutional encoder–decoder architecture. The encoder gradually downsamples the input, trading spatial resolution for increasingly abstract and global representations. The decoder then upsamples these representations back to the original resolution to produce the output. A defining feature of the U-Net is the presence of skip connections that pass feature maps directly from the encoder to the decoder at matching spatial resolutions.

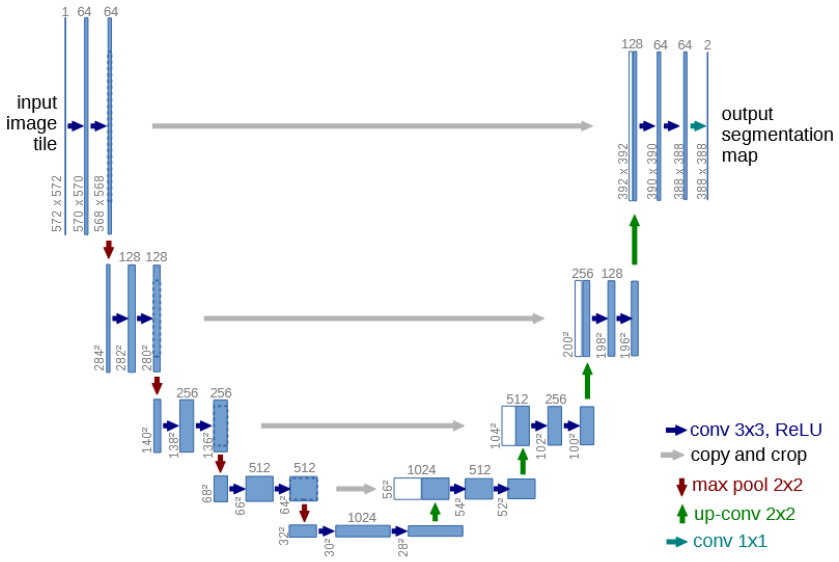



**Figure 19:** Canonical U-Net architecture introduced by Ronneberger et al [15]. The symmetric encoder–decoder structure and skip connections enable multi-scale feature integration.

These connections ensure that fine-grained spatial information, which is often degraded by repeated downsampling, remains available during reconstruction.

Figure 19 illustrates the canonical U-Net architecture introduced by Ronneberger et al., highlighting the symmetric encoder–decoder structure and the skip connections that link corresponding resolution levels.

In modern diffusion models, the U-Net is typically constructed from residual blocks. Each residual block includes a residual connection that adds the block input to the output of a learned transformation.

Using the notation of Section 2.0.2, a residual block can be written as

$$h^{(l+1)} = h^{(l)} + f\left(h^{(l)}, t\right), \tag{90}$$

where $h^{(l)}$ denotes the feature representation (feature map) at depth $l$ in the network, and $f$ represents the transformation applied within a residual block, consisting of a sequence of operations such as convolutions, normalization, and nonlinearities. The diffusion timestep $t$ enters as a conditioning variable inside this transformation. In Figure 19, each residual block corresponds to one such mapping from $h^{(l)}$ to $h^{(l+1)}$.

The mapping defined by Eq. (90) constitutes a residual (skip) connection operating locally within each block. This should be distinguished from the long-range skip connections in the U-Net architecture, which connect encoder and decoder layers at matching spatial resolutions and are also shown in Figure 19. Rather than learning a completely new representation, each residual block learns a correction to the identity mapping. If this correction is small, the representation passes through the block largely unchanged.

This design has important implications for optimization. During backpropagation, the gradient satisfies

$$\delta^{(l)} = \delta^{(l+1)} \left( I + \frac{\partial f}{\partial h^{(l)}} \right), \tag{91}$$

which ensures that an identity path for gradient flow is always present, even when the derivative of $f$ is small. Residual connections therefore mitigate vanishing gradients and make deep networks easier to train. In diffusion models, this added stability is particularly important, as the noise-prediction network must be optimized consistently across many diffusion timesteps. Intuitively, residual connections encourage each layer to refine the current representation rather than reconstruct it from scratch [6].

The skip connections in a U-Net serve a different but complementary role. Instead of modifying the transformation within a layer, they transfer intermediate feature maps from the encoder to the decoder by concatenation at the same spatial resolution. In Figure 19, these skip connections correspond to the horizontal links between encoder and decoder blocks at matching depths. Schematically, this can be written as

$$h^{(r)} = \mathrm{concat}\big(h^{(r)}_{\mathrm{dec}},\, h^{(r)}_{\mathrm{enc}}\big), \tag{92}$$

where $h^{(r)}_{\mathrm{enc}}$ denotes the activation of the encoder at level $r$ and $h^{(r)}_{\mathrm{dec}}$ denotes the corresponding decoder activation before concatenation [15].

While the encoder and decoder features share the same spatial dimensions, the encoder activations retain high-frequency spatial details that are often lost during downsampling. By concatenating these features, the decoder gains access to both global context and local structure when forming its predictions.

The distinction between residual connections and U-Net skip connections is therefore straightforward. A residual connection combines the input of a block with the output of a learned transformation through addition,

$$h^{(l)} = [1], \quad f(h^{(l)}, t) = [5] \quad \rightarrow \quad h^{(l+1)} = [6], \tag{93}$$

so that the existing representation is updated by a learned correction rather than being replaced.

This additive structure encourages each block to refine the current features while maintaining a direct identity path through the network.

A U-Net skip connection combines encoder and decoder representations through concatenation [15],

$$h_{\text{enc}}^{(r)} = [1], \quad h_{\text{dec}}^{(r)} = [5] \quad \rightarrow \quad h^{(r)} = [5, 1]. \tag{94}$$

making encoder features explicitly available to the decoder alongside its own activations, rather than merging them additively. While this operation does not prevent information loss fully caused by earlier downsampling, it allows the decoder to reuse higher-resolution features when forming its predictions. Subsequent layers can then learn how strongly each component should influence the final output.

## 2.5 Training

### 2.5.1 Model and Architecture

All experiments in this thesis are conducted on the MNIST dataset [12], a standard benchmark for image-based machine learning. The dataset consists of 60,000 grayscale images of handwritten digits (0–9), each with spatial resolution $28 \times 28$. Due to its limited complexity and single-channel structure, MNIST does not require the depth or capacity typically needed for large-scale RGB image datasets. Since the goal of this thesis is to analyze the bias–variance behavior of diffusion models rather than to optimize generative performance, MNIST provides a controlled and computationally efficient setting that is well suited to this study.

The model used throughout the experiments is a U-Net architecture. While U-Nets are standard in diffusion models [9], the version employed here is intentionally simplified. Training and sampling diffusion models is computationally demanding, and MNIST does not necessitate a high-capacity network. Accordingly, the architecture uses three resolution levels rather than the four or more commonly found in larger-scale models. This choice reduces computational cost while retaining sufficient expressive power for the denoising task.

The U-Net is trained to approximate the noise-prediction function $\epsilon_\theta(x_t, t)$, at each step of the diffusion process. What the network is expected to do changes over time: at early timesteps the input is heavily corrupted by noise and the model must remove large-scale noise, while at later timesteps the noise level is lower and more fine-grained corrections are needed.

The noisy input $x_t$ alone does not fully indicate which of these situations the network is in, since similar-looking inputs can arise at different diffusion steps. The network therefore needs to be explicitly informed of the current timestep [9].

Although the timestep $t$ is just a single scalar, using it directly is not very helpful for learning. Instead, $t$ is mapped to a higher-dimensional representation using sinusoidal time embeddings, following the positional encoding scheme introduced for Transformers [19]. This gives the network a more useful representation of time and allows timestep information to interact with feature activations throughout the U-Net. In particular, the time embedding is injected into each residual block, ensuring that all stages of the network are conditioned on the current diffusion step [9]. This enables the model to smoothly adapt its denoising behavior as the diffusion process progresses.

An overview of the architecture is shown in Figure 20. The input image $x_t$ is first processed by a $3 \times 3$ convolution that maps the single grayscale channel to 32 feature channels, defining the base width of the network. This constitutes the first of three resolution levels, with channel widths scaled by factors $1, 2$ and 4. At each resolution level, two residual blocks are applied, followed by downsampling via a stride-2 convolution, which halves the spatial resolution while increasing the number of channels.

At the lowest resolution, the network applies a set of bottleneck residual blocks before transitioning to the decoder. Upsampling is performed using transposed convolutions with kernel size $4 \times 4$ and stride 2. This choice provides uniform coverage of the output grid and helps mitigate checkerboard artifacts that can arise from uneven overlap in transposed convolutions [13].

After each upsampling step, feature maps from the corresponding encoder level are concatenated via U-Net skip connections and processed by the residual blocks at that resolution.

The internal structure of a residual block is shown in Figure 21. Each block begins with group normalization, which helps keep activations on a stable scale during training. This is particularly important in diffusion models, where batch sizes are typically small and batch normalization becomes unreliable. [20].

Given activations $a \in \mathbb{R}^{\mathcal{B} \times C \times H \times W}$, where $\mathcal{B}$ is the batch size, $C$ the number of channels, and $H \times W$ the spatial dimensions, group normalization divides the channel dimension into $G$ disjoint groups of equal size and normalizes each group independently for each sample.

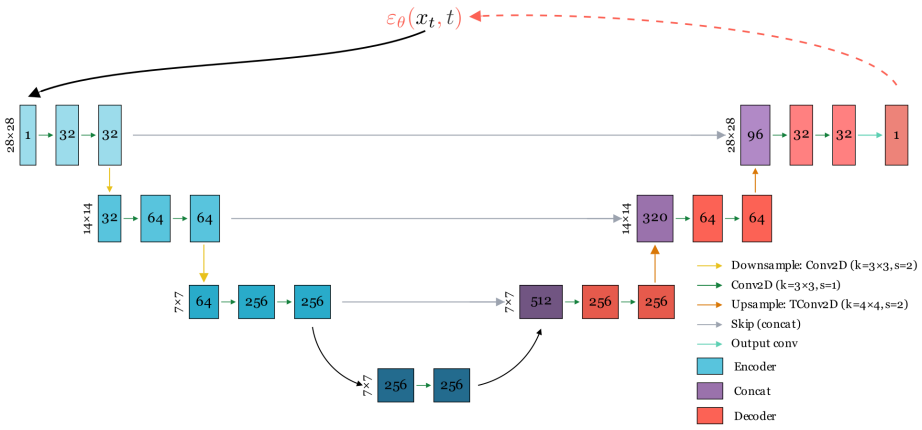



**Figure 20:** Simplified U-Net architecture used for noise prediction in the diffusion model, consisting of three resolution levels with residual blocks, downsampling and upsampling paths, and skip connections.

For a given sample and group $G$, normalization is performed over all spatial locations and channels within that group. The mean and variance are computed as

$$\mu_{x_t,G} = \frac{1}{m}\sum_{i=1}^{m} a_i, \quad \sigma^2_{x_t,G} = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu_{x_t,G})^2, \tag{95}$$

where $m = \frac{C}{G}HW$ is the total number of activations in the group, since each group contains $\frac{C}{G}$ channels and each channel spans $H \times W$ spatial locations. The normalized activations are then given by

$$\hat{a}_i = \frac{a_i - \mu_{x_t,G}}{\sqrt{\sigma^2_{x_t,G} + \xi}} \tag{96}$$

with $\xi > 0$ a small constant added for numerical stability. Throughout the network, $G = 16$ groups are used, except in the final layer where $G = 4$.

The normalized activations are then passed through a learnable per-channel affine transformation to compensate for the possible loss of representational capacity introduced by normalization,

$$y_i = \gamma_{C(i)}\hat{a}_i + \beta_{C(i)}, \tag{97}$$

where $\gamma_C, \beta_C \in \mathbb{R}$ are trainable scale and shift parameters for channel $C$, and $C(i)$ denotes the channel index associated with activation $a_i$ [20].

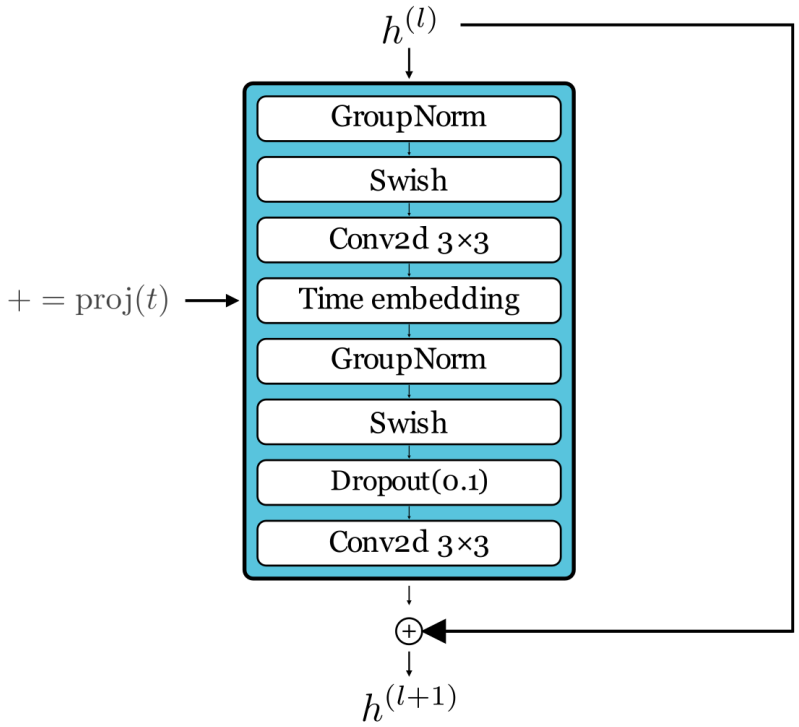



**Figure 21:** Structure of a residual block used throughout the U-Net, including group normalization, Swish activation, convolutional layers, timestep conditioning, and a residual connection.

These affine parameters allow the network to recover or suppress features after normalization and ensure that group normalization does not restrict the expressiveness of the model.

Following normalization, the Swish activation function is applied [14],

$$\text{Swish}(x) = x\sigma(x), \quad \sigma(x) = \frac{1}{1+e^{-x}}. \tag{98}$$

which allows small negative activations to propagate through the network. Compared to ReLU, this results in smoother gradients and reduces the risk of inactive units, as illustrated in Figure 22.

After the Swish activation and the first convolution, each residual block incorporates explicit timestep conditioning through the time embedding shown in Figure 21, [9]. The scalar timestep $t$ is first mapped to a high-dimensional embedding vector using the sinusoidal encoding described earlier.

This embedding is then passed through a learned linear projection and added to the intermediate feature maps within the block, corresponding to the "+ =" operation shown in the figure.

Dropout with rate 0.1 is applied within the residual blocks for regularization. Each block concludes with a residual connection, implemented as either the identity mapping or a $1 \times 1$ convolution when the number of channels changes. Together, these design choices yield a compact and stable architecture that is well matched to the denoising task on MNIST.

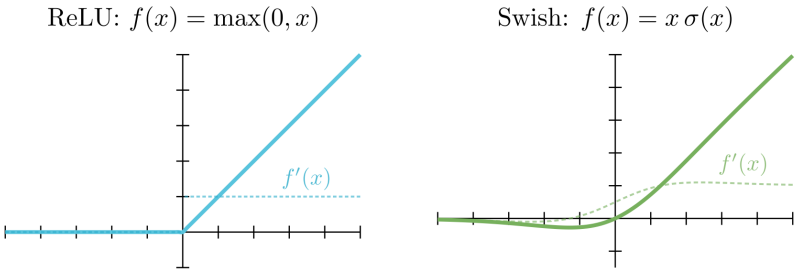



**Figure 22:** Comparison of ReLU and Swish activation functions, illustrating the smoother behavior of Swish for negative inputs and its non-zero gradients [14].

### 2.5.2 Diffusion model configuration

The diffusion model is configured according to the standard DDPM framework introduced in Section 2.3 [9]. A finite Markov chain is defined with a fixed, non-trainable forward process $q$ and a trainable reverse process $p_\theta$ modelled by a U-net. The number of diffusion steps is set to $T = 1000$, which is sufficient to ensure stable training on the MNISTdataset.

Noise injection in the forward process is governed by a fixed linear variance schedule [9],

$$\beta_t \in [10^{-4}, 0.02], \quad t = 1, \ldots, T. \tag{99}$$

This schedule determines the amount of Gaussian noise added at each diffusion step and fully specifies the forward corruption process.

Although the forward process is formally defined as a sequence of conditional transitions $q(x_t|x_{t-1})$, explicit sequential simulation of the Markov chain is unnecessary during training. Instead, noisy samples $x_t$ are obtained directly from the closed-form marginal distribution $q(x_t|x_0)$, which can be sampled in a single step using a draw $\epsilon \sim \mathcal{N}(0, \mathcal{I})$ [9]. This approach is mathematically equivalent to iterating the forward process, but is significantly more efficient and allows arbitrary diffusion timesteps to be sampled independently.

The model is trained using the simplified mean squared error objective derived in the previous section,

$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}[||\epsilon - \epsilon_\theta(x_t, t)||^2]. \tag{100}$$

which trains the network to predict the noise realization added at timestep $t$.

The full diffusion procedure used throughout this thesis is summarized in two components: the training procedure for the reverse process and the sampling algorithm used for generation. During training, the timestep $t$ is sampled uniformly from $\{1, \ldots, T\}$ to ensure that the noise-prediction network is trained evenly across all diffusion steps, preventing bias toward particular noise levels [9].

**Algorithm 1:** Training (Reverse process) [9]

**repeat**
- $x_0 \sim q(x_0)$
- $t \sim \text{Uniform}(\{1, \ldots, T\})$
- $\epsilon \sim \mathcal{N}(0, \mathcal{I})$
- take gradient descent step on

$$\nabla_\theta ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t)||_2^2$$

**until** convergence;

**Algorithm 2:** Sampling with the backward process [9]

**Require:** trained noise predictor $\epsilon_\theta(x_t, t)$

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

**for** $t = T, T-1, \ldots, 1$ **do**
- $z \sim \mathcal{N}(0, I)$
- $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_q(t)\, z$

**end for**

**return** $x_0$

### 2.5.3 Training configuration

Before training, the MNIST dataset was split into training and validation subsets using an 80/20 partition, resulting in 48,000 training images and 12,000 validation images. All images are grayscale and are therefore represented as 2-dimensional matrices. Pixel intensities are linearly rescaled to the range $[-1, 1]$, which is standard practice in diffusion models and improves numerical stability when Gaussian noise is injected during the forward process.

Training is performed using mini-batches of size 16 to avoid computing gradients over the full dataset at each update. Each training step therefore consists of sampling 16 independent data points $x_0$ together with 16 independently sampled diffusion timesteps $t$. The model is trained for 25 epochs, where one epoch corresponds to a full pass over the 48,000 training samples, yielding

$$\frac{48,000}{16} = 3000$$

parameter updates per epoch. Prior to each epoch, the training data are shuffled to ensure varied mini-batch composition, while the validation set is kept fixed and unshuffled.

Due to the stochastic nature of diffusion training and the noise present in the objective, an exponential moving average (EMA) of the model parameters is maintained throughout training [9]. Let $\theta^{(k)}$ denote the parameters at iteration $k$. The EMA parameters $\hat{\theta}^{(k)}$ are updated according to

$$\hat{\theta}^{(k)} = \lambda_{EMA}\hat{\theta}^{(k-1)} + (1 - \lambda_{EMA})\theta^{(k)} \tag{101}$$

where $\lambda_{\text{EMA}}$ controls the decay rate. EMA-smoothed parameters are used for validation and sampling, as they typically yield more stable training behavior and higher-quality samples.

Model parameters are optimized using the Adam optimizer [11], an adaptive variant of stochastic gradient descent that maintains exponential moving averages of both gradients and their elementwise squares. Denoting by $\nabla_k \in \mathbb{R}^d$ the stochastic gradient of the loss with respect to the parameter vector $\theta$ at iteration $k$, the first- and second-moment estimates are computed as

$$s_k = \rho_1 s_{k-1} + (1 - \rho_1)\nabla_k, \quad r_k = \rho_2 r_{k-1} + (1 - \rho_2)\nabla_k \odot \nabla_k, \tag{102}$$

where $s_k, r_k \in \mathbb{R}^d$ are vectors of the same dimension as $\theta$, $\odot$ denotes elementwise multiplication, and $\rho_1, \rho_2 \in (0, 1)$ are decay rates.

Bias-corrected estimates are then given by

$$\hat{s}_k = \frac{s_k}{1 - \rho_1}, \quad \hat{r}_k = \frac{r_k}{1 - \rho_2}, \tag{103}$$

and parameters are updated according to

$$\theta_{k+1} = \theta_k - \eta \frac{\hat{s}_k}{\sqrt{\hat{r}_k} + \xi}, \tag{104}$$

with learning rate $\eta$ and a small constant $\xi > 0$ included for numerical stability.

All hyperparameter choices used throughout the experiments—including dataset configuration, preprocessing, batching, optimization, and EMA settings are summarized in Table 1.

**Table 1:** Training configuration used for all experiments.

| Category | Parameter | Value |
|---|---|---|
| Data | Dataset | MNIST |
| | Training samples | 48,000 |
| | Validation samples | 12,000 |
| Preprocessing | Image size | $28 \times 28$ |
| | Channels | 1 (grayscale) |
| | Range | $[-1, 1]$ |
| Batching | Batch size | 16 |
| | Training shuffle | Enabled |
| | Validation shuffle | Disabled |
| Optimization | Optimizer | Adam |
| | Learning rate $\eta$ | $2 \times 10^{-4}$ |
| | Adam decay rates $(\rho_1, \rho_2)$ | $(0.9, 0.999)$ |
| | Adam stability constant $\xi$ | $10^{-8}$ |
| Training | Epochs | 25 |
| | Smoothing | EMA |
| | EMA decay rate $\lambda_{EMA}$ | 0.9999 |

## 2.6 Validation methods

### 2.6.1 Inception Score

Evaluating the quality of samples produced by a generative model is inherently challenging, as there is no ground-truth notion of a "correct" output. Unlike supervised learning, generated samples cannot be directly compared to reference targets, and qualitative inspection by human eyes therefore remains common. To complement such subjective evaluation, Salimans et al. [17] introduced the Inception Score (IS), a heuristic metric based on the behavior of a pretrained image classifier applied to generated samples.

Let $x$ denote a generated image and let $y$ denote the class label predicted by the classifier. The classifier defines a conditional distribution $p(y|x)$ over labels given an image, and a marginal distribution $p(y) = \mathbb{E}_x[p(y|x)]$ over labels across the generated dataset. The intuition underlying the Inception Score has two components. First, individual generated images should correspond to recognizable objects, which is reflected in confident classifier predictions. Formally, this corresponds to a low-entropy conditional distribution $p(y|x)$ (high fidelity).

Second, the generator should produce a diverse collection of images, so that the marginal distribution $p(y)$ has high entropy (high diversity). A desirable generative model therefore produces samples that are both individually sharp and collectively diverse.

The Inception Score encodes both requirements through the Kullback–Leibler divergence between $p(y \mid x)$ and $p(y)$. A high score is obtained only when individual samples induce sharp label distributions while the aggregate of generated samples covers multiple semantic classes.

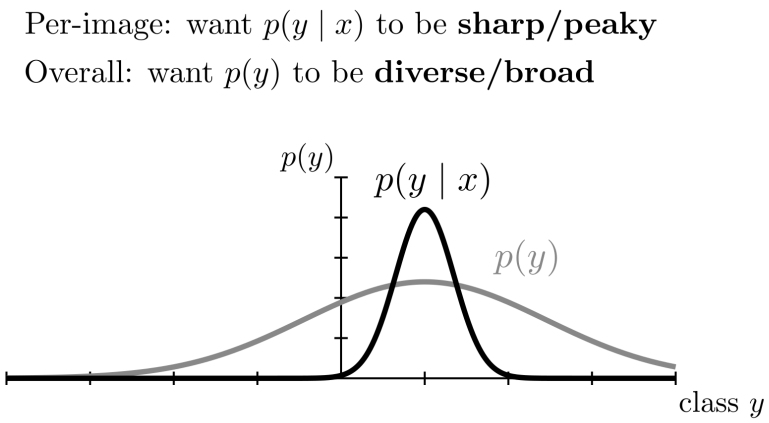



**Figure 23:** Intuition behind the Inception Score. High values arise when per-sample label distributions $p(y \mid x)$ are sharp while the marginal distribution $p(y)$ across samples is diverse.

Figure 23 illustrates this intuition schematically: for a fixed image $x$, the conditional distribution $p(y \mid x)$ produced by the classifier is typically concentrated on a small set of labels, whereas the marginal distribution $p(y)$, obtained by averaging over many generated images, is more broadly spread.

Let $g$ denote the generator and let $p_{\mathrm{cl}}(y \mid x)$ denote the output of a pretrained classifier, commonly taken to be the Inception network [17]. The Inception Score is defined as

$$\mathrm{IS} = \exp\left(\mathbb{E}_{x \sim p^{(g)}}\left[D_{\mathrm{KL}}\left(p_{\mathrm{cl}}(y \mid x) \,||\, p_{\mathrm{cl}}^{(g)}(y)\right)\right]\right), \tag{105}$$

where

$$p_{\mathrm{cl}}^{(g)}(y) := \int p^{(g)}(x')\, p_{\mathrm{cl}}(y \mid x')\, \mathrm{d}x' \tag{106}$$

is the marginal label distribution induced by the generator through the classifier.

In practice, the expectation over $x \sim p_g$ is approximated by Monte Carlo averaging over a finite set of generated samples. To gain analytical insight, however, it is useful to expand the expectation in closed form. Since $Y$ is discrete with $K$ possible classes, we obtain

$$\log \mathrm{IS} = \int p^{(g)}(x) \left[ \sum_{y=1}^{K} p_{\mathrm{cl}}(y \mid x) \log \frac{p_{\mathrm{cl}}(y \mid x)}{p_{\mathrm{cl}}^{(g)}(y)} \right] \mathrm{d}x \tag{107}$$

$$= \underbrace{\int p^{(g)}(x) \sum_{y=1}^{K} p_{\mathrm{cl}}(y \mid x) \log p_{\mathrm{cl}}(y \mid x) \, \mathrm{d}x}_{(A)} - \underbrace{\int p^{(g)}(x) \sum_{y=1}^{K} p_{\mathrm{cl}}(y \mid x) \log p_{\mathrm{cl}}^{(g)}(y) \, \mathrm{d}x}_{(B)} . \tag{108}$$

The first term can be rewritten as

$$(A) = \sum_{y=1}^{K} \int p^{(g)}(x) \, p_{\mathrm{cl}}(y \mid x) \log p_{\mathrm{cl}}(y \mid x) \, \mathrm{d}x \tag{109}$$

$$= \sum_{y=1}^{K} \int p_{X,Y}(x, y) \log p_{\mathrm{cl}}(y \mid x) \, \mathrm{d}x \tag{110}$$

$$= -H(Y \mid X), \tag{111}$$

while the second term becomes

$$(B) = \sum_{y=1}^{K} \left( \int p^{(g)}(x) \, p_{\mathrm{cl}}(y \mid x) \, \mathrm{d}x \right) \log p_{\mathrm{cl}}^{(g)}(y) \tag{112}$$

$$= \sum_{y=1}^{K} p_{\mathrm{cl}}^{(g)}(y) \log p_{\mathrm{cl}}^{(g)}(y) \tag{113}$$

$$= -H(Y). \tag{114}$$

Combining these expressions yields

$$\log \mathrm{IS} = H(Y) - H(Y \mid X) = I(X; Y), \tag{115}$$

showing that the Inception Score admits a mutual information interpretation, where $X$ denotes generated images and $Y$ the corresponding classifier labels.

This interpretation immediately implies bounds on the score. Since mutual information is nonnegative and bounded above by the entropy of $Y$, we have

$$0 \leq I(X; Y) = H(Y) - H(Y \mid X) \leq H(Y) \leq \log K. \tag{116}$$

With only $K$ classes, the predicted label can convey at most $\log K$ bits of information.

Interpreting entropy as a measure of uncertainty clarifies the behavior of the metric. A small conditional entropy $H(Y \mid X)$ indicates confident predictions for individual samples, which is often associated with visually sharp and coherent images. A large marginal entropy $H(Y)$ indicates that predicted labels are well distributed across classes, reflecting diversity rather than mode collapse. The Inception Score therefore rewards generators that produce images that are easy to classify on a per-sample basis while remaining diverse overall, without explicitly measuring fidelity relative to the true data distribution.

Finally, it is important to emphasize that the Inception Score is a heuristic rather than a principled validation measure for generative models. As noted by Salimans et al. [17], the score is computed solely on generated samples and depends entirely on the behavior of a pretrained classifier. It therefore reflects classifier confidence rather than semantic correctness, and it does not directly compare the generated distribution to the true data distribution. Moreover, the common use of an Inception network pretrained on the ImageNet dataset introduces a bias toward ImageNet-like semantics, limiting the reliability of the metric for datasets with substantially different visual characteristics or label structures.

### 2.6.2 Fréchet Inception Distance (FID)

The objective of a generative model is not merely to produce visually plausible samples, but to learn the underlying data distribution from which those samples are drawn. For this reason, evaluating generative models is most naturally framed as a distribution-level problem rather than as an assessment of individual outputs.

Metrics such as the Inception Score primarily reflect image sharpness and diversity, but they do not directly quantify how closely the generated samples match real data.

A more informative evaluation should therefore compare the distributions of real and generated samples directly. This motivates the use of distribution-based measures such as the Fréchet Inception Distance (FID) [7], which addresses some of the limitations of the Inception Score. In particular, unlike IS, FID explicitly compares real and generated data and is sensitive to both sample quality and mode coverage.

This comparison is not performed in the raw image space, but in a learned feature space defined by a deep convolutional network. Specifically, both real and generated images are passed through a pretrained Inception network, and activations from a hidden layer close to the output are used as feature representations.

This representation is used because it compares images based on their visual content rather than exact pixel values, which aligns better with human perception.

Let $P_r$ and $P_g$ denote the real and generated data distributions in a learned feature space. FID approximates these distributions by multivariate Gaussians, $\mathcal{N}_r(\mu_r, \Sigma_r)$ and $\mathcal{N}_g(\mu_g, \Sigma_g)$, whose parameters are estimated from empirical means and covariances. The FID score is defined as

$$\text{FID} = ||\mu_r - \mu_g||_2^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\right). \tag{117}$$

This expression corresponds to the squared 2-Wasserstein distance between two Gaussian distributions [3, 4]. Intuitively, it measures the cost of transforming the generated feature distribution into the real one. The availability of a closed-form expression makes FID computationally convenient, though it is exact only when the feature distributions are themselves Gaussian.

Minimizing FID therefore enforces agreement between the first two moments of the distributions, that is, $\mu_g \approx \mu_r$ and $\Sigma_g \approx \Sigma_r$. While matching these moments is necessary, it is not sufficient to guarantee full distributional alignment. Differences in higher-order structure, such as multimodality or localized density variations, may still persist.

This limitation is illustrated in Figure 24. Although the real and generated distributions have similar means, indicated by the green line connecting $\mu_r$ and $\mu_g$, and comparable overall spread, they differ in shape. In particular, the generated distribution contains an additional localized region of high density that is absent from the real data. Despite being visually prominent, this region carries relatively little probability mass and therefore has only a minor effect on the mean and covariance.

Figure 25 shows the same distributions after Gaussian approximation. Since these approximations retain only first- and second-order moments, the localized discrepancy is largely smoothed out. As a result, the Gaussian ellipsoids appear much more similar than the original distributions.

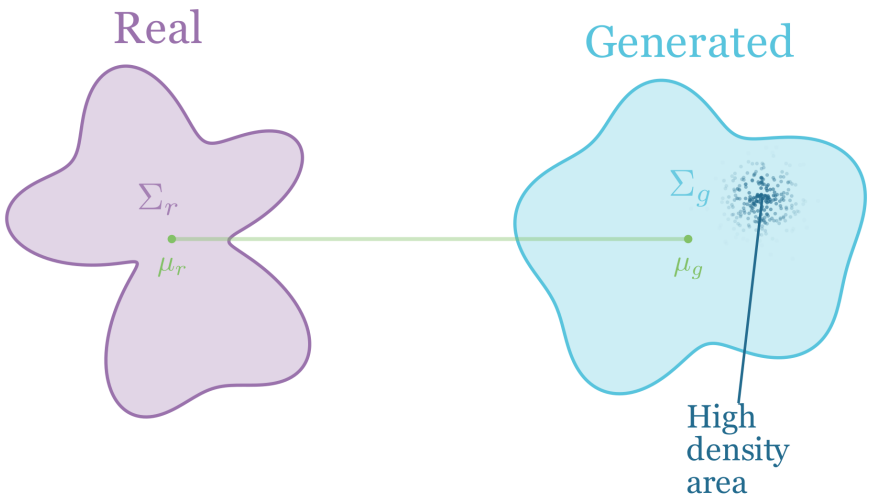



**Figure 24:** Example illustrating a limitation of moment-based metrics: two distributions may share identical means and covariances while differing in higher-order structure.

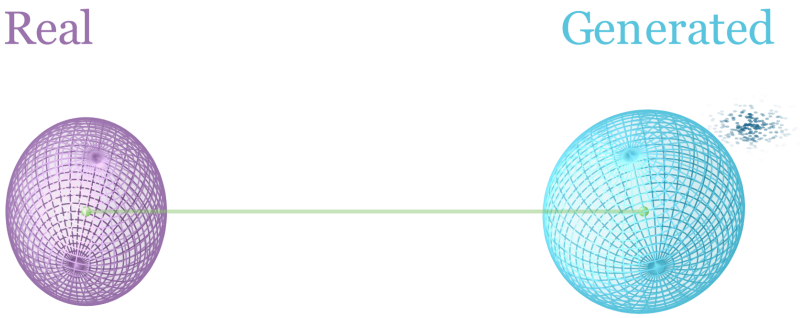



**Figure 25:** Gaussian approximations of real and generated feature distributions, highlighting that FID compares only first- and second-order moments in feature space.

To make the computation explicit, consider a simple two-dimensional example with feature activations

$$a_g = \begin{pmatrix} 9.13 & 0.38 \\ 6.94 & 0.05 \end{pmatrix} \qquad a_r = \begin{pmatrix} 0.09 & 0.27 \\ 0.77 & 0.00 \end{pmatrix} \tag{118}$$

where $a_g$ denotes generated features and $a_r$ real features. The corresponding empirical means and covariances are

$$\mu_r = \begin{pmatrix} 8.04 \\ 0.22 \end{pmatrix} \qquad \mu_g = \begin{pmatrix} 0.43 \\ 0.14 \end{pmatrix} \tag{119}$$

and

$$\Sigma_r = \begin{pmatrix} 1.21 & 0.18 \\ 0.18 & 0.03 \end{pmatrix} \qquad \Sigma_g = \begin{pmatrix} 0.12 & -0.05 \\ -0.05 & 0.02 \end{pmatrix} \tag{120}$$

The mean mismatch term is

$$\mu_r - \mu_g = \begin{pmatrix} 8.04 - 0.43 \\ 0.22 - 0.14 \end{pmatrix} = \begin{pmatrix} 7.61 \\ 0.08 \end{pmatrix} \tag{121}$$

giving

$$||\mu_r - \mu_g||_2^2 = 7.61^2 + 0.08^2 = 57.9185. \tag{122}$$

The remaining term involves the matrix square root of the covariance product. Since $\Sigma_r \Sigma_g$ is not guaranteed to be symmetric or positive semidefinite, the square root may be ill-defined or numerically unstable. The standard remedy is to use the symmetric formulation

$$\mathrm{Tr}\left((\Sigma_r \Sigma_g)^{1/2}\right) = \mathrm{Tr}\left((\Sigma_r^{1/2} \Sigma_g \Sigma_r^{1/2})^{1/2}\right) \tag{123}$$

which is guaranteed to be symmetric and positive semidefinite.

Eigen-decomposition yields

$$\Sigma_r^{1/2} = U \Lambda^{1/2} U^\top \approx \begin{pmatrix} 1.0891 & 0.1541 \\ 0.1541 & 0.0791 \end{pmatrix}, \tag{124}$$

and

$$\Sigma_r^{1/2} \Sigma_g \Sigma_r^{1/2} = \begin{pmatrix} 1.0891 & 0.1541 \\ 0.1541 & 0.0791 \end{pmatrix} \begin{pmatrix} 0.12 & -0.05 \\ -0.05 & 0.02 \end{pmatrix} \begin{pmatrix} 1.0891 & 0.1541 \\ 0.1541 & 0.0791 \end{pmatrix} \tag{125}$$

$$\approx \begin{pmatrix} 0.1260 & 0.0148 \\ 0.0148 & 0.0017 \end{pmatrix}. \tag{126}$$

Taking the square root again gives

$$\left(\Sigma_r^{1/2} \Sigma_g \Sigma_r^{1/2}\right)^{1/2} = \begin{pmatrix} 0.3526 & 0.0415 \\ 0.0415 & 0.0059 \end{pmatrix}. \tag{127}$$

The resulting FID score is therefore

$$\begin{aligned}
\text{FID} &= ||\mu_r - \mu_g||_2^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r^{1/2}\Sigma_g\Sigma_r^{1/2}\right)^{1/2}\right) && (128)\\
&= ||\mu_r - \mu_g||_2^2 + \text{Tr}\left(\Sigma_r + \Sigma_g\right) - 2\text{Tr}\left(\left(\Sigma_r^{1/2}\Sigma_g\Sigma_r^{1/2}\right)^{1/2}\right) && (129)\\
&= 57.9185 + 1.38 - 2(0.3585)\\
&= 58.5815.
\end{aligned}$$

In this example, the FID score is dominated by the large mean mismatch, indicating substantial bias and poor fidelity: the generated features are centered far from the real ones. In addition, the trace of the generated covariance is much smaller than that of the real covariance, meaning that the generated distribution has substantially less overall spread. If the distributions were centered at the same location, this reduction would correspond to insufficient diversity or mode collapse.

The trace of the individual covariances captures only the total variance of each distribution, not how that variance is distributed across directions. The square-root interaction term accounts for this directional structure and shows that the limited variability in the generated distribution is not only smaller in magnitude, but also misaligned with the principal directions of real data variability. Taken together, these effects point to a form of structural mismatch between the generated and real distributions in this example, rather than excessive dispersion. This interpretation relies on the Gaussian approximation underlying FID; when this approximation is poor, important discrepancies in higher-order structure may remain undetected.

A low FID score therefore indicates agreement at the level of second-order moments, but it does not guarantee that the full data distribution has been captured. Despite this limitation, FID remains a widely used and practical metric. By comparing real and generated samples in a learned feature space, it provides a concise and computationally efficient summary of distributional similarity, particularly when interpreted alongside qualitative inspection or complementary evaluation measures.

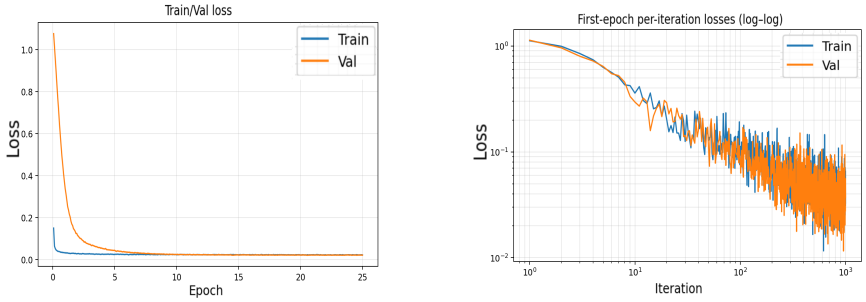
# 3 Results

## 3.1 Training Behavior and Sample Evolution

As discussed in Section 2.3.3, training a DDPM can be viewed as a regression problem, which naturally admits a bias–variance decomposition [9]. In this setting, bias reflects systematic error in the learned noise predictor $\epsilon_\theta(x_t, t)$, leading to consistent under- or over-removal of noise at different diffusion steps. Variance, by contrast, reflects sensitivity of the learned denoising function to the particular training data and is typically associated with unstable or noisy denoising dynamics.

Figure 26 shows the training and validation loss curves for the DDPM. Both curves decrease rapidly at the beginning of training and then transition into a slowly converging plateau. This pattern indicates that most error reduction occurs early, with later epochs primarily refining an already learned denoising rule rather than introducing qualitatively new improvements. Throughout training, the training and validation losses remain closely aligned, with no sustained divergence.

A small gap between training and validation loss is visible during the initial phase, but this gap closes quickly. There is also no indication of a widening separation at later epochs. In classical supervised learning, such divergence is commonly associated with increasing variance and overfitting. Its absence here suggests that variance remains well controlled and that the dominant source of error throughout training is residual bias rather than variance.




**(a)** **(b)**

**Figure 26:** Training and validation loss curves for the DDPM. (a): loss per epoch showing rapid early convergence followed by a stable plateau with minimal train–validation gap. (b): training and validation loss during the first 1,000 iterations (log–log scale), illustrating fast initial error reduction and increased stochastic fluctuations at later iterations.

Figure 26b provides a closer look at the first epoch by plotting per-iteration losses on log–log axes. Even within a single epoch, the loss decreases by several orders of magnitude, confirming that early optimization steps account for a substantial fraction of learning. As training progresses and the loss becomes smaller, higher-frequency fluctuations become more pronounced. Importantly, these fluctuations appear with comparable magnitude in both the training and validation curves.

This within-epoch noise should not be interpreted as evidence of overfitting or increasing model variance. Instead, it reflects stochasticity inherent to the diffusion training objective, arising from random timestep sampling and random noise realizations. Because the same fluctuations are present in both training and validation losses, they are algorithmic rather than data-specific. This suggests that the variance observed during DDPM training is primarily driven by stochastic sampling of diffusion steps, rather than by sensitivity to individual training examples in the classical sense.

Beyond the initial convergence phase, no secondary regimes, instabilities, or divergences are observed. Once the loss reaches a plateau, the training and validation curves remain nearly indistinguishable. While this behavior supports the presence of a stable convergence region, it also limits what can be inferred from loss curves alone, as the remaining loss may reflect irreducible noise in the regression target or intrinsic limitations of the model.

Figure 27 shows the mean squared noise-prediction residual as a function of the diffusion timestep $t$,

$$\text{Residual}(t) = \mathbb{E}_{x_0,\epsilon}[||\epsilon - \epsilon_\theta(x_t, t)||^2], \tag{130}$$

which measures the expected prediction error at a fixed diffusion step.

Unlike the training loss, which averages error across all diffusion steps, this diagnostic reveals how prediction error varies along the diffusion trajectory. The residual curve shows that the error is large at small timesteps, followed by a gradual decay as $t$ increases. The error is highest for early diffusion steps, decreases steadily for intermediate values of $t$, and approaches a low, nearly constant level at large $t$. Prediction error is therefore not evenly distributed across the reverse process, but is heavily concentrated in the low-noise regime.

This pattern follows directly from the structure of the forward diffusion process. Recall that

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, \tag{131}$$

where $\bar{\alpha}_t$ decreases monotonically with $t$. For large $t$, $\bar{\alpha}_t \approx 0$, and $x_t$ is dominated by Gaussian noise. In this regime, the denoising task reduces to predicting noise from an input that already resembles noise, which is comparatively straightforward and leads to consistently small residuals.

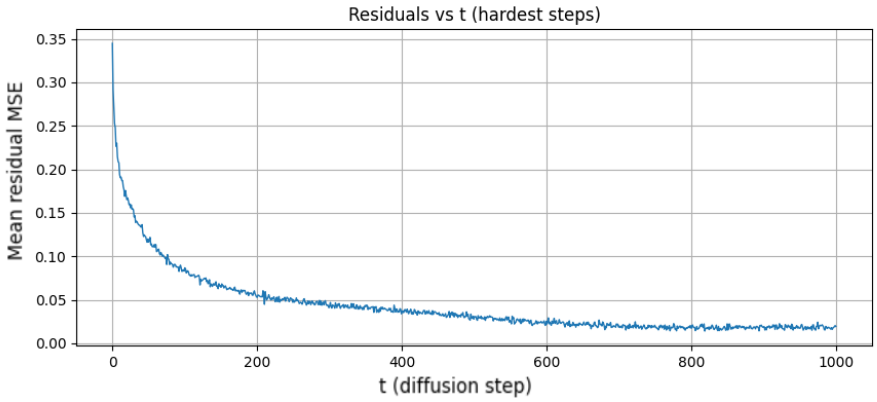



**Figure 27:** Mean squared noise-prediction residual as a function of diffusion timestep $t$. Prediction error is concentrated at small timesteps and decays toward a low-error plateau for large $t$, indicating that denoising is substantially more difficult in low-noise regimes.

For small $t$, by contrast, $\bar{\alpha}_t \approx 1$, and $x_t$ is close to the clean image with only a small, sample-specific noise component. Here the model must both recognize the underlying image structure and estimate a subtle perturbation. This makes the regression problem substantially more demanding and leads to larger prediction errors.

From another perspective, inputs at large $t$ are statistically similar across samples, which results in relatively homogeneous gradients during training. At small $t$, the inputs retain strong image-dependent structure, so the required denoising corrections vary more across samples. Accurate denoising in this regime therefore requires fine-grained, image-specific adjustments, which naturally produces larger and more variable residuals.

This explains why a large fraction of the total prediction error is concentrated at small $t$, even though timesteps are sampled uniformly during training. High-$t$ denoising is learned quickly and accounts for much of the early loss reduction, while later optimization is increasingly dominated by the more difficult low-$t$ regime. This observation is consistent with the findings of Ho et al., who reported that explicitly reweighting the loss to emphasize small $t$ does not reliably improve sample quality.

It is worth pointing out that this diagnostic has clear limitations. Because the residuals are averaged over samples and training iterations, the curve does not indicate whether the large errors at small $t$ arise uniformly across the dataset or are driven by a subset of difficult examples. The plot therefore identifies where error is concentrated along the diffusion trajectory, but not how that error is distributed across data points or noise realizations.

Figure 28 shows generated samples at several training checkpoints, illustrating how sample quality changes as training progresses. The sequence highlights how improvements in the noise-prediction objective translate into progressively more structured outputs.

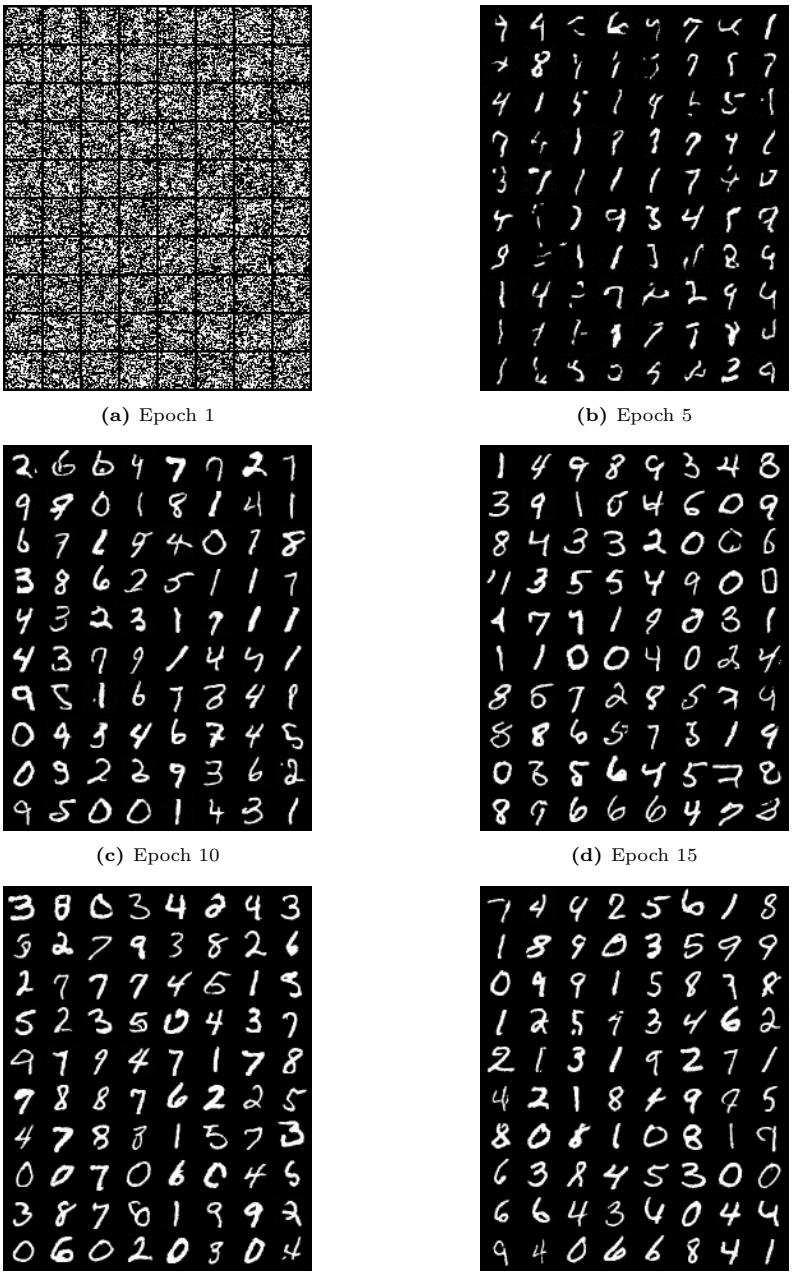
At epoch 1, the generated samples are almost entirely noise. The reverse process is not yet informative enough to reconstruct meaningful structure. This is notable because the training and validation losses decrease most rapidly during this early phase. The lack of recognizable structure shows that early reductions in the noise-prediction loss do not imply that the model has learned the conditional relationships required for generation. At this stage, the estimator $\epsilon_\theta(x_t, t)$ remains strongly biased across timesteps, especially in low-noise regimes where structure-dependent corrections are necessary.

By epoch 5, coarse digit-like patterns begin to appear, although most samples are still difficult to identify. Rough spatial organization emerges, such as approximate stroke placement, but shapes remain fragmented and unstable. This suggests that the model has started to capture broad structural regularities while fine-scale details are still poorly estimated.

Between epochs 10 and 15, most samples become visually legible. Digit identities are generally recognizable, and global shapes are more consistent. Characteristic stroke layouts for digits such as “2”, “6”, and “8” appear reliably. At the same time, local features, such as stroke thickness, curvature, and intersections remain uneven. This indicates that variability across samples is reduced earlier in training than systematic errors at finer spatial scales. In other words, dispersion decreases before residual bias in detailed denoising is fully resolved.

From epochs 20 to 25, visual changes are minor, indicating saturation in observable sample quality. No collapse toward identical or near-identical samples is observed at any point. Diversity is preserved throughout training, with remaining variation appearing mainly as stylistic differences rather than major structural defects. This suggests that variance remains controlled even as the model continues to refine its predictions.

Despite convergence of the training and validation losses to low values, generated samples still exhibit imperfections, including distorted strokes and occasional ambiguous digit identities. This underscores a limitation of loss-based diagnostics: low average prediction error does not guarantee perfect sample fidelity. From the samples alone, it is not possible to determine whether the remaining errors are spread evenly across diffusion steps or concentrated in specific parts of the trajectory. The samples therefore illustrate how structure and variability evolve during training, but they do not fully characterize the learned data distribution.



**(a)** Epoch 1

**(b)** Epoch 5

**(c)** Epoch 10

**(d)** Epoch 15

**(e)** Epoch 20

**(f)** Epoch 25

**Figure 28:** Generated MNIST samples at selected training epochs. Sample quality improves steadily from unstructured noise to recognizable digits, with most qualitative gains occurring early in training and only marginal refinements at later epochs.

## 3.2 Evaluation of Fidelity and Diversity

In this subsection, model performance is evaluated using the Inception Score (IS) and the Fréchet Inception Distance (FID) [7, 17]. Both metrics rely on a pretrained neural network: IS is computed from classifier output probabilities, while FID compares statistics of feature representations extracted from an intermediate layer.

IS and FID are often computed using Inception v3 pretrained on ImageNet, but this choice is inappropriate for MNIST due to the large mismatch in image resolution, channel structure, and semantic content. Instead, both metrics are computed using a classifier trained specifically on MNIST, based on a reduced ResNet-18 architecture. For FID, feature vectors are extracted from the ResNet layer corresponding to the role of the pool3 layer in Inception v3.

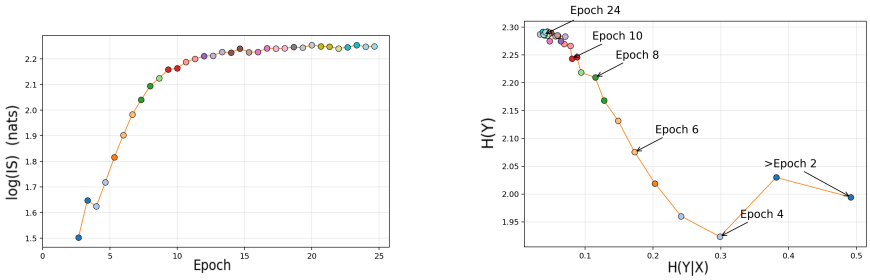
All evaluations use exponential moving average (EMA) parameters rather than the raw model weights, as EMA smoothing is known to produce more stable and higher-quality samples in diffusion models [9]. IS and FID are evaluated every 2,000 training iterations. Because EMA estimates are unreliable early in training due to initialization effects, the first 8,000 iterations are excluded from evaluation. Each reported IS value is averaged over five independent runs to reduce variability from sampling noise.

Figure 29a shows the evolution of the Inception Score over training epochs. The score increases rapidly during the early stages of training and then levels off. Most of the improvement occurs within the first 10 epochs, after which the score fluctuates around a stable value with only minor changes. This pattern aligns with the qualitative progression observed in generated samples and suggests that further optimization yields limited gains once a stable generative behavior has been established.

To examine this trend more closely, Figure 29b shows the trajectory of the model in the entropy plane defined by the marginal entropy $H(Y)$ and conditional entropy $H(Y|X)$, as discussed in Section 2.6.1.

Early in training, the model occupies a region with high conditional entropy and lower marginal entropy, indicating that individual samples are ambiguous to the classifier and that predicted class frequencies are uneven. As training proceeds, the trajectory moves toward lower $H(Y|X)$ and higher $H(Y)$, reflecting sharper classifier predictions for individual samples together with more balanced class coverage.

Most of this movement occurs during the early and middle stages of training, consistent with the steep rise in the IS curve. After roughly 10 epochs, the trajectory concentrates in a narrow region characterized by high marginal entropy and low conditional entropy.



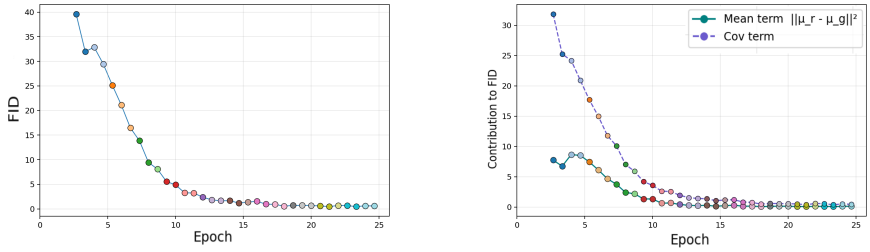

**(a)** **(b)**

**Figure 29:** Inception Score (IS) evaluation over training. (a): log(Inception Score) as a function of training epochs, showing rapid early improvement followed by saturation. (b): trajectory in the entropy plane ($H(Y)$, $H(Y|X)$), illustrating increasing class confidence and sustained class-level diversity.

This means that we see convergence to a regime in which both sample-level clarity and class-level diversity change little with additional training. Notably, no drift toward low marginal entropy is observed as conditional entropy decreases, suggesting that improvements in sample clarity are not achieved by reducing diversity across classes.

The IS curve and entropy-plane analysis show that improvements in sample fidelity occur early and then saturate, while class-level diversity remains stable throughout training. Although IS does not directly measure bias or variance, these results are consistent with earlier findings: systematic denoising error is reduced early, and there is no evidence of instability or collapse at later stages of training.

Figure 30 shows how the Fréchet Inception Distance (FID) evolves over training. Figure 30a reports the total FID as a function of training epochs, while Figure 30b breaks the score into contributions from mean matching and covariance matching in feature space.




**(a)** **(b)**

**Figure 30:** Fréchet Inception Distance (FID) evaluation over training. (a): total FID as a function of epochs, exhibiting a sharp early decrease and subsequent stabilization. (b): decomposition of FID into mean-matching and covariance-matching terms, showing that early improvements are dominated by reductions in feature-space covariance mismatch.

Compared to the Inception Score, FID changes more gradually over the course of training and does not display large relative swings. This difference follows from how the two metrics are defined. The Inception Score depends exponentially on entropy differences, so linear changes in its logarithm translate into multiplicative changes in the score itself. FID, by contrast, is based on squared distances between feature-space means and covariances, and therefore reflects absolute reductions in distributional mismatch.

The total FID drops sharply during the early stages of training and then settles into a narrow range. As with the IS results, most of the improvement occurs before roughly epoch 10, with subsequent epochs producing only modest reductions. This pattern shows that the model aligns the generated and real feature distributions early in training, after which further optimization yields limited additional improvement.

The decomposition of FID clarifies the source of this early progress. The covariance term decreases rapidly, while the mean term falls more slowly. This means that generated samples quickly contract into a region of feature space with a spread similar to that of the data, reducing excessive variability. Matching of the average feature representation proceeds at a slower pace, reflecting more gradual refinement of global structure.

Neither the covariance term nor the total FID rises at later epochs. This behavior rules out a late-stage loss of diversity or a drift away from the data distribution. Instead, training reaches a stable regime in which the overall spread of generated samples is controlled early, and remaining discrepancies are tied to smaller, incremental adjustments in feature alignment.

Viewed together with the Inception Score results, the FID curves reinforce a consistent picture: the largest improvements in distribution-level agreement occur early in training, while later epochs mainly preserve this agreement without introducing instability or collapse.

# 4 Conclusions

This thesis examined diffusion-based generative models, with a focus on Denoising Diffusion Probabilistic Models (DDPMs), to investigate how the classical bias–variance framework applies in a generative setting. By exploiting the regression interpretation of DDPM training, bias and variance were studied through their effects on sample fidelity, diversity, and stability, rather than through prediction error alone.

Empirically, the results show that residual bias is the main factor limiting sample quality. While fidelity improves over training, generated samples remain slightly over-smoothed compared to the data, consistent with incomplete noise removal. This points to systematic denoising error as the dominant source of imperfection. In contrast, no clear signs of high variance are observed. Generated samples remain diverse, stable across runs, and free from memorization or collapse. This suggests that the learned denoising function is not overly sensitive to the training data. Variance therefore does not act as a competing force to bias in improving fidelity; instead, its role appears to be maintaining stable and diverse sampling behavior.

More fundamentally, the absence of classical overfitting patterns may reflect a structural difference between DDPMs and standard supervised models. In DDPM training, the regression target is a noise realization that is resampled independently at every iteration. Even for the same clean image and timestep, the target changes from step to step, producing an effectively unbounded set of labels. This stochasticity makes direct memorization unlikely and causes training and validation objectives to remain statistically similar. As a result, common indicators of overfitting such as a widening gap between training and validation loss may not arise, even for expressive models.

Consistent with this interpretation, no evidence of overfitting is observed in either the loss curves or the generated samples. One possible explanation is that the chosen model architecture is insufficiently complex to enter a high-variance regime, implying that training remains in an underfitting-dominated phase. Another possibility is that the training duration was too short for overfitting effects to emerge.

The results indicate that the way bias and variance appear in DDPMs differs from what is typically seen in supervised regression. In the diffusion setting, limitations in sample quality are mainly linked to systematic errors in noise prediction, which restrict how accurately noise can be removed at certain diffusion steps. In these experiments, this behavior appears as consistently higher prediction error at small timesteps. In contrast, effects commonly associated with high variance, such as unstable training or growing discrepancies between training and validation loss, are not observed.

Figure 26 shows that the training and validation losses remain closely aligned throughout training, indicating stable optimization without signs of overfitting.

27 further shows that prediction error is concentrated at small timesteps rather than increasing over time. This suggests that the dominant limitation is persistent denoising error in low-noise regimes, rather than instability in the learned reverse process.

This distinction matters when interpreting commonly used generative metrics such as IS and FID. These metrics summarize multiple aspects of generative performance in a single score, but they do not distinguish between errors caused by systematic limitations in noise removal and those arising from unstable training dynamics.

Future work could build on this analysis in several ways. First, additional evaluation metrics, such as precision–recall for generative models, could be used to better capture variance-related effects that are hard to isolate using IS and FID alone [16]. Second, since variance-related issues do not appear to be a major factor in the current experiments, it would be useful to explore noise schedules other than the linear schedule used in this thesis. Different schedules may change how difficult denoising is at different timesteps and could affect training dynamics, convergence, and the trade-off between image quality and diversity. Finally, the high computational cost of DDPM sampling remains a practical challenge. Using faster or reduced sampling schedules could allow for longer training runs and more extensive experiments, making it easier to observe overfitting effects and study variance in diffusion models more directly.

In conclusion, this thesis shows that although DDPM training can be written as a regression problem, the classical bias–variance picture from supervised learning does not apply directly. Bias appears primarily as persistent inaccuracies in noise removal that limit sample fidelity, while variance is more closely tied to the stability of the reverse process and the preservation of diversity. This separation helps explain why low training loss and stable validation behavior can coexist with imperfect samples. Understanding these differences is important for interpreting generative performance and for reasoning about generalization in diffusion models as they are scaled and applied to more complex settings.

# 5 Appendix

## 5.1 Derivation of closed form expression for $\mu_q(x_t, x_0)$ and $\sigma_q^2(t)$.

Starting from Eq. (37), the conditional density $q(x_{t-1} \mid x_t, x_0)$ is proportional to the product of the two Gaussian factors

$$q(x_{t-1} \mid x_t, x_0) \;\propto\; \mathcal{N}\left(x_t; \sqrt{\alpha_t}\, x_{t-1}, (1-\alpha_t)\mathcal{I}\right) \mathcal{N}\left(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\, x_0, (1-\bar{\alpha}_{t-1})\mathcal{I}\right).$$

To apply Eq. (38), we rewrite the first factor as a Gaussian function of $x_{t-1}$ so that both terms are Gaussians in the same variable. We have that

$$\mathcal{N}\left(x_t; \sqrt{\alpha_t}, x_{t-1}, (1-\alpha_t)\mathcal{I}\right) \propto \exp\left(-\frac{1}{2(1-\alpha_t)}||x_t - \sqrt{\alpha_t}x_{t-1}||^2\right).$$

If we expand the norm we find that the exponent becomes

$$-\frac{1}{2(1-\alpha_t)}(\alpha_t||x_{t-1}||^2 - 2\sqrt{\alpha_t}x_t^\top x_{t-1} + ||x_t||^2).$$

We can drop the $||x_t||^2$ term since it is constant in $x_{t-1}$, leaving

$$\begin{aligned} q(x_t|x_{t-1}) &\propto \exp\left(-\frac{1}{2(1-\alpha_t)}(\alpha_t||x_{t-1}||^2 - 2\sqrt{\alpha_t}x_t^\top x_{t-1})\right) \\ &\propto \exp\left(-\frac{\alpha_t}{2(1-\alpha_t)}(||x_{t-1}||^2 - 2\frac{1}{\sqrt{\alpha_t}}x_t^\top x_{t-1})\right). \end{aligned}$$

We can now complete the square in $x_{t-1}$

$$||x_{t-1}||^2 - 2\frac{1}{\sqrt{\alpha_t}}x_t^\top x_{t-1} = \left|\left|x_{t-1} - \frac{1}{\sqrt{\alpha_t}}x_t\right|\right|^2 - \left|\left|\frac{1}{\sqrt{\alpha_t}}x_t\right|\right|^2.$$

Once again, the last term is constant in $x_{t-1}$ so

$$q(x_t|x_{t-1}) \propto \exp\left(-\frac{\alpha_t}{2(1-\alpha_t)}\left|\left|x_{t-1} - \frac{1}{\sqrt{\alpha_t}}x_t\right|\right|^2\right).$$

This has the Gaussian form $\exp\left(-\frac{1}{2\sigma^2}||x_{t-1} - \mu||^2\right)$ up to constants independent of $x_{t-1}$. Matching coefficients yields

$$q(x_t|x_{t-1}) \propto \mathcal{N}\left(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}x_t, \frac{1-\alpha_t}{\alpha_t}\mathcal{I}\right).$$

The second factor is already expressed as a Gaussian density in $x_{t-1}$, so its parameters are immediate. Consequently, define

$$\begin{aligned} \mu_1 &:= \frac{1}{\sqrt{\alpha_t}}x_t, & \sigma_1^2 &:= \frac{1-\alpha_t}{\alpha_t}, \\ \mu_2 &:= \sqrt{\bar{\alpha}_{t-1}}x_0, & \sigma_2^2 &:= 1 - \bar{\alpha}_{t-1}. \end{aligned}$$

Now we can apply Eq. (38). We have that

$$\begin{aligned}\sigma_q^2(t) &= \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2} \\ &= \frac{\frac{1-\alpha_t}{\alpha_t}(1-\bar{\alpha}_{t-1})}{\frac{1-\alpha_t}{\alpha_t}+(1-\bar{\alpha}_{t-1})}. \end{aligned} \tag{*}$$

We focus on the denominator and find that

$$\begin{aligned}\frac{1-\alpha_t}{\alpha_t}+(1-\bar{\alpha}_{t-1}) &= \frac{1-\alpha_t}{\alpha_t}+\frac{\alpha_t(1-\bar{\alpha}_{t-1})}{\alpha_t} \\ &= \frac{(1-\alpha_t)+\alpha_t(1-\bar{\alpha}_{t-1})}{\alpha_t} \\ &= \frac{1-\alpha_t+\alpha_t-\alpha_t\bar{\alpha}_{t-1}}{\alpha_t} \\ &= \frac{1-\alpha_t\bar{\alpha}_{t-1}}{\alpha_t}.\end{aligned}$$

Using that $\bar{\alpha}_t = \alpha_t\bar{\alpha}_{t-1}$, we find that

$$\frac{1-\alpha_t\bar{\alpha}_{t-1}}{\alpha_t} = \frac{1-\bar{\alpha}_t}{\alpha_t}.$$

Plug this into (*)

$$\begin{aligned}\sigma_q^2(t) &= \frac{\frac{1-\alpha_t}{\alpha_t}(1-\bar{\alpha}_{t-1})}{\frac{1-\alpha_t}{\alpha_t}+(1-\bar{\alpha}_{t-1})} \\ &= \frac{\frac{1-\alpha_t}{\alpha_t}(1-\bar{\alpha}_{t-1})}{\frac{1-\bar{\alpha}_t}{\alpha_t}} \\ &= \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t},\end{aligned}$$

which is what we have in Eq. (40). For $\mu_q(x_t, x_0)$ we have

$$\begin{aligned}\mu_q &= \frac{\mu_1\sigma_2^2+\mu_2\sigma_1^2}{\sigma_2^2+\sigma_1^2} \\ &= \frac{\frac{1-\bar{\alpha}_{t-1}}{\sqrt{\alpha_t}}x_t+\sqrt{\bar{\alpha}_{t-1}}\frac{1-\alpha_t}{\alpha_t}x_0}{\frac{1-\bar{\alpha}_t}{\alpha_t}} \\ &= \frac{\alpha_t}{1-\bar{\alpha}_t}\left(\frac{1-\bar{\alpha}_{t-1}}{\sqrt{\alpha_t}}x_t+\sqrt{\bar{\alpha}_{t-1}}\frac{1-\alpha_t}{\alpha_t}x_0\right).\end{aligned}$$

Factor out $\alpha_t$ and we find Eq. (39)

$$\begin{aligned}\mu_q(x_t, x_0) &= \frac{\alpha_t}{1-\bar{\alpha}_t}\left(\frac{1-\bar{\alpha}_{t-1}}{\sqrt{\alpha_t}}x_t + \sqrt{\bar{\alpha}_{t-1}}\frac{1-\alpha_t}{\alpha_t}x_0\right) \\ &= \frac{1}{1-\bar{\alpha}_t}(\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0) \\ &= \frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t}.\end{aligned}$$

## 5.2 ELBO Derivation.

We begin from the negative evidence lower bound

$$-\mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right].$$

To simplify notation, we drop the expectation operator and work directly with the log-density term. All expressions are understood to be taken under $q(x_{1:T}|x_0)$.

Recall that the joint distributions factorize as

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \qquad q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}).$$

Substituting these into the log ratio yields

$$\begin{aligned}
-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} &= -\log\left[p(x_T)\prod_{t=1}^{T} p(x_{t-1}|x_t) - \prod_{t=1}^{T} q(x_t|x_{t-1})\right] \\
&= -\log p(x_T) - \sum_{t\geq 1}^{T} \log p_\theta(x_{t-1}|x_t) + \sum_{t\geq 1}^{T} \log q(x_t|x_{t-1}),
\end{aligned}$$

which can be rewritten as

$$-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} = -\log p(x_T) - \sum_{t\geq 1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}.$$

At $t = 1$ the variable $x_0$ is observed rather than latent, so the corresponding ELBO contribution reduces to the likelihood term $-\log p_\theta(x_0|x_1)$ instead of a KL divergence. We therefore separate the $t = 1$ term, leaving only terms involving latent variables $x_{t-1}$ for $t > 1$ which can be expressed as KL divergences:

$$-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} = -\log p(x_T) - \sum_{t>1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} - \log \frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}$$

The marginal transitions $q(x_t|x_{t-1})$ are intractable, while the conditional reverse distributions $q(x_{t-1}|x_t, x_0)$ admit closed-form expressions. Applying Bayes' rule to the forward process conditioned on $x_0$ yields

$$q(x_t|x_{t-1}) = \frac{q(x_{t-1}|x_t, x_0)q(x_t|x_0)}{q(x_{t-1}|x_0)}.$$

Substituting this into the sum over $t > 1$ gives

$$
\begin{aligned}
-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} &= -\log p(x_T) - \sum_{t>1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{\left(\frac{q(x_{t-1}|x_t,x_0)q(x_t|x_0)}{q(x_{t-1}|x_0)}\right)} - \log \frac{p_\theta(x_0|x_1)}{q(x_1|x_0)} \\
&= -\log p(x_T) - \sum_{t>1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)} - \sum_{t>1}^{T} \log \frac{q(x_{t-1}|x_t)}{q(x_t|x_0)} - \log \frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}.
\end{aligned}
$$

Now, if we focus on the second sum we can see that

$$
\begin{aligned}
\sum_{t>1}^{T} \log \frac{q(x_{t-1}|x_t)}{q(x_t|x_0)} &= \log \frac{q(x_1|x_0)}{q(x_2|x_0)} + \log \frac{q(x_2|x_0)}{q(x_3|x_0)} + \cdots + \log \frac{q(x_{T-1}|x_0)}{q(x_T|x_0)} \\
&= \log q(x_1|x_0) - \log q(x_2|x_0) + \log q(x_2|x_0) \\
&\quad - \log q(x_3|x_0) + \cdots + \log q(x_{T-1}|x_0) - \log q(x_T|x_0) \\
&= \log q(x_1|x_0) - \log q(x_T|x_0) \\
&= \log \frac{q(x_1|x_0)}{q(x_T|x_0)}.
\end{aligned}
$$

Substituting this back yields

$$
\begin{aligned}
-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} &= -\log p(x_T) - \sum_{t>1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)} - \sum_{t>1}^{T} \log \frac{q(x_{t-1}|x_t)}{q(x_t|x_0)} - \log \frac{p_\theta(x_0|x_1)}{q(x_1|x_0)} \\
&= -\log p(x_T) - \sum_{t>1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)} - \log \frac{q(x_1|x_0)}{q(x_T|x_0)} - \log \frac{p_\theta(x_0|x_1)}{q(x_1|x_0)} \\
&= -\log p(x_T) - \sum_{t>1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)} - \log q(x_1|x_0) \\
&\quad + \log q(x_T|x_0) - \log p_\theta(x_0|x_1) + \log q(x_1|x_0) \\
&= -\log \frac{p(x_T)}{q(x_T|x_0)} - \sum_{t>1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)} - \log p_\theta(x_0|x_1).
\end{aligned}
$$

Taking expectations with respect to $q(x_{1:T}|x_0)$, we obtain

$$
\begin{aligned}
&\mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] = \\
&= \mathbb{E}_{q(x_{1:T}|x_0)}\left[D_{KL}(q(x_T|x_0)||p(x_T)] + \sum_{t>1} D_{KL}[q(x_{t-1}|x_t,x_0))||p_\theta(x_{t-1}|x_t)] - \log p_\theta(x_0|x_1)\right].
\end{aligned}
$$

# References


[1] Christopher M. Bishop and Hugh Bishop. *Deep Learning: Foundations and Concepts*. Springer, 2024. DOI: 10.1007/978-3-031-45468-4.

[2] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG]. URL: https://arxiv.org/abs/2105.05233.

[3] D. C. Dowson and B. V. Landau. "The Fréchet Distance between Multivariate Normal Distributions". In: *Journal of Multivariate Analysis* 12.3 (1982), pp. 450–455. DOI: 10.1016/0047-259X(82)90077-X.

[4] Matthias Gelbrich. "On a Formula for the $L^2$ Wasserstein Metric between Measures on Euclidean and Hilbert Spaces". In: *Mathematische Nachrichten* 147 (1990), pp. 185–203. DOI: 10.1002/mana.19901470121.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016. URL: https://www.deeplearningbook.org.

[6] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[7] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG]. URL: https://arxiv.org/abs/1706.08500.

[8] Catherine F. Higham, Desmond J. Higham, and Peter Grindrod. *Diffusion Models for Generative Artificial Intelligence: An Introduction for Applied Mathematicians*. 2023. arXiv: 2312.14977 [cs.LG]. URL: https://arxiv.org/abs/2312.14977.

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG]. URL: https://arxiv.org/abs/2006.11239.

[10] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML]. URL: https://arxiv.org/abs/1312.6114.

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[12] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[13] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and Checkerboard Artifacts". In: *Distill* (2016). DOI: 10.23915/distill.00003.

[14] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE]. URL: https://arxiv.org/abs/1710.05941.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV]. URL: https://arxiv.org/abs/1505.04597.

[16] Mehdi S. M. Sajjadi et al. *Assessing Generative Models via Precision and Recall*. 2018. arXiv: 1806.00035 [stat.ML]. URL: https://arxiv.org/abs/1806.00035.

[17] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG]. URL: https://arxiv.org/abs/1606.03498.

[18] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. 2020. arXiv: 1907.05600 [cs.LG]. URL: https://arxiv.org/abs/1907.05600.

[19] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[20] Yuxin Wu and Kaiming He. *Group Normalization*. 2018. arXiv: 1803.08494 [cs.CV]. URL: https://arxiv.org/abs/1803.08494.

[21] Aston Zhang et al. *Dive into Deep Learning*. Cambridge University Press, 2023. URL: https://d2l.ai.