Diffusion Models in Depth: From a Theoretical and a Practical Perspective

Gregory Igor Sedykh
Département d'Informatique
Université de Genève
Carouge GE, Switzerland
gregory.sedykh@etu.unige.ch

Acknowledgement

I would like to thank my supervisor Prof. Stéphane Marchand-Maillet for his feedback and help throughout the writing of this Bachelor's Thesis.

Abstract

This report aims to provide a detailed and comprehensive overview of the current state of diffusion models, a type of generative model that has gained popularity over the past few years. The foundations of the theory behind diffusion models, particularly Denoising Diffusion Probabilistic Models (DDPMs), are seen in depth along with the most important improvements made to them. To illustrate the theory and to be able to visualise the performance of diffusion models, a Jupyter Notebook is provided with a practical implementation of a simple DDPM using PyTorch. Overall, the reader will be able to understand the theory, visualise the model, be able to implement it, comprehend the improvements and the reasons behind them and observe the difference in performance between the different models so as to have an understanding of the current progress made in the field of diffusion models.

Contents

1 Introduction **5**

2 Denoising Diffusion Probabilistic Models **6**

2.1 Forward Process 6

2.2 Reverse Process 7

2.2.1 Variational Lower Bound 8

2.2.2 Simplified training objective 13

2.2.3 Model architecture 14

3 Score-based formulation **17**

3.1 Score matching 17

3.2 Sampling 18

3.3 Noise Conditional Score Networks 19

3.4 Equivalence between DDPM and NCSN 20

4 Improvements upon DDPMs **21**

4.1 Improved likelihood 21

4.2 Denoising Diffusion Implicit Models (DDIMs) 23

4.3 Guidance 24

4.3.1 Classifier guidance 25

4.3.2 Classifier-free guidance 25

5 Further development **27**

5.1 Cascaded Diffusion Models 27

5.2 Latent Diffusion Models 28

5.3 Diffusion Transformer 28

6 Implementation **30**

7 Conclusion **32**

A Convolution of two Gaussian distributions **33**

1 Introduction

Diffusion models have gained widespread popularity since 2020, as models such as *DALL-E*, *Stable Diffusion* [1, 2] and *Midjourney* have proven to be capable of generating high-quality images given a text prompt. Furthermore, OpenAI's recent announcement of *Sora* has shown that diffusion models have now also become highly capable of generating minute long high-definition videos from a text prompt [3].

These models date back to 2015, where the idea of a diffusion model appeared, based on diffusion processes used in thermodynamics [4]. Denoising Diffusion Probabilistic Models (DDPMs) were a development of the original diffusion probabilistic model introduced in 2015 [5]. Subsequently, OpenAI improved upon the original DDPMs which did not have ideal log likelihoods [5] while also using less forward passes and therefore speeding up the sampling process [6].
The most recent progress done by OpenAI has allowed their diffusion models to obtain better metrics and better sample quality than Generative Adversarial Networks (GANs) which were previously considered the state-of-the-art in image generation [7].

The fairly recent apparition of diffusion models means not only that there is still a lot to be discovered about them, but also that progress is being made rapidly. The theory behind diffusion models was mainly founded when Ho et al. [5] introduced their DDPMs in 2020, but many improvements have been made upon their work since then.
To understand what these models are and how they work, it is crucial to understand how DDPMs were developed, what choices were made when developing them and why these choices were made, as well as why changes were made to the original model and how they were made.

This report aims to provide an overview of the theory behind diffusion models, as well as a practical guide on how to implement a simple diffusion model using PyTorch, in order to compare what the theory shows us and what the practical implementation gives us.

2 Denoising Diffusion Probabilistic Models

A **diffusion model** is a generative model that consists of two Markov chains, one forward and one reverse. Given an input (e.g. an image), the forward process will destroy the information in the image by gradually adding Gaussian noise at each step of the process [5].
The reverse process' objective is to "invert" the forward process and learn how to do so, starting from a noisy uninformative image and step-by-step, estimating the structure of the information that was destroyed in the image at each step and regenerating it until the first step is reached, where we should obtain the original image [5].

2.1 Forward Process

The forward process is a Markov process that starts from the original image x_0 and adds Gaussian noise during T steps which results in a more noisy image x_t [5].
At each step $t \in [1, T]$, the noise added has variance $\beta_t \in \{\beta_1, \dots, \beta_T\}$.
This process is defined by q which is a probability distribution that takes as input an image x_t and outputs its likelihood:

$$q : \mathbb{R}^D \to [0, 1] \tag{2.1}$$

where D is the data dimensionality (e.g. for a 64×64 RGB image, $D = 64 \times 64 \times 3$).
Pixels in the image x_t are assumed to be mutually independent conditionally to the previous timestep. This is reflected by the fact that the covariance matrix $(\beta_t I)$ is diagonal.

Formally, we obtain the following Markov process:

$$q(x_1, \dots, x_T | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1}) \tag{2.2}$$

where:

$$q(x_t | x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I\right) \tag{2.3}$$

Equation 2.3 gives us a single step forward: given the previous image x_{t-1}, x_t is obtained by sampling a D-dimensional Gaussian distribution with mean $\sqrt{1 - \beta_t} x_{t-1}$ and variance $\beta_t I$.
Equation 2.2 gives us the full forward process from the original image x_0 to the final image x_T.

The reparametrisation trick says that for a univariate Gaussian distribution where $z \sim p(z|x, \mu) = \mathcal{N}\left(x, \sigma^2\right)$, a valid reparametrisation would be $z = x + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$ is just noise [8].
In other words, sampling z conditionally to x from a Gaussian distribution is equivalent to getting z as x with Gaussian noise of variance σ^2 added to it. In this case, we can see that the image x_t becomes more random because of the Gaussian noise ϵ_t added to x_{t-1} at each step, since we are sampling:

$$\begin{aligned} x_t &= q(x_t | x_{t-1}) && (2.4) \\ &= \sqrt{1 - \beta_t} x_{t-1} + \sqrt{\beta_t} I \epsilon_t \qquad \epsilon_t \sim \mathcal{N}_D(0, I) && (2.5) \end{aligned}$$

It is important to note that as $T \to \infty$ and with a correct choice of β_t, x_T will become a sample of an isotropic Gaussian distribution ($\mathcal{N}(0, I)$) [6, 4].
This is important for the reverse process, as it will allow us to take a sample $x_T \sim \mathcal{N}(0, I)$ and reverse the forward process to obtain the original image x_0 (however this cannot be done so simply, as seen in section 4) [6].

Ho et al. [5] use the cascade of the reparametrisation trick to be able to sample x_t at any arbitrary step t of the forward process.
This property is due to the fact that the space of Gaussian distributions is closed under convolution. In other words, the addition of two samples from Gaussian distributions is the same as obtaining a sample from the convolution of the initial distributions (see Appendix A).

Let $\alpha_t = 1 - \beta_t$, then:

$$\begin{aligned} x_t &\sim \mathcal{N}\left(\sqrt{1 - \beta_t} x_{t-1}, \beta_t I\right) \\ x_t &= \sqrt{1 - \beta_t} x_{t-1} + \sqrt{\beta_t} \epsilon_{t-1} \qquad \epsilon_{t-1} \sim \mathcal{N}_D(0, I) \\ &= \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \epsilon_{t-1} \end{aligned}$$

From this, we can apply it again to x_{t-1} and obtain:

$$x_{t-1} = \sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon_{t-2}$$

Therefore, we get:

$$x_t = \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{\alpha_t (1-\alpha_{t-1})}\epsilon_{t-2} + \sqrt{1-\alpha_t}\epsilon_{t-1}$$

We can write:

$$x_t = \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \mathcal{N}(0, \alpha_t (1-\alpha_{t-1}) I) + \sqrt{1-\alpha_t}\epsilon_{t-1} \tag{2.6}$$
$$= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \mathcal{N}(0, \alpha_t (1-\alpha_{t-1}) I) + \mathcal{N}(0, (1-\alpha_t)) \tag{2.7}$$

The convolution of the two Gaussian distributions $\mathcal{N}(\mu_1, \sigma_1^2)$ and $\mathcal{N}(\mu_2, \sigma_2^2)$ gives us a new Gaussian distribution with mean $\mu_1 + \mu_2$ and variance $\sigma_1^2 + \sigma_2^2$ (see Appendix A):

$$x_t = \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \mathcal{N}(0, (\alpha_t (1-\alpha_{t-1}) + 1 - \alpha_t) I) \tag{2.8}$$

Which finally gives us:

$$x_t = \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}}\bar{\epsilon}_{t-2}$$

Where $\epsilon_{t-1}, \epsilon_{t-2}, \ldots \sim \mathcal{N}(0, I)$ and $\bar{\epsilon}_{t-2}$ is the convolution of the noise distributions ϵ_{t-1} and ϵ_{t-2}.

We can recursively apply this backwards until x_0.
Let $\bar{\alpha}_t = \prod_{i=0}^{t} \alpha_i$:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \tag{2.9}$$

This will give us a way to directly compute a noisy image x_t at step t of the forward process, given the original image x_0 (we ignore the noise $\epsilon \sim \mathcal{N}_D(0, I)$):

$$q(x_t|x_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t} x_0, (1-\bar{\alpha}_t) I\right) \tag{2.10}$$

We now also know that $1 - \bar{\alpha}_t$ is the equivalent variance of the noise added to x_0 to obtain x_t during the forward process [6].

Choosing the variances β_t is an important step when developing the forward process.
Ho et al. [5] chose to use a linear schedule starting from $\beta_1 = 10^{-4}$ and increasing linearly until $\beta_T = 0.02$ in order for them to be small compared to the pixel values that were in $[-1, 1]$.
However we will see in section 4 that this is not the best choice, as this destroys the images too quickly closer to the end of the process while adding little to sample quality, which made the model sub-optimal for 64×64 and 32×32 images [6].

Finally, Ho et al. [5] chose $T = 1000$ in order to be able to compare their model with Sohl et al.'s [4] model, which also used $T = 1000$ for most image experiments.
We will again see in section 4 that this many steps can make the sampling slow and that there exist better choices [6].

2.2 Reverse Process

As mentioned previously, with a correct choice of β_t, x_T will become the sample of an isotropic Gaussian distribution over $\mathbb{R}^D$ as $T \to \infty$ [6, 4].
This should mean that we can take a sample $x_T \sim \mathcal{N}_D(0, I)$ and reverse the forward process to obtain the original image x_0. However, this means that we need to use the entire dataset in order to figure out $q(x_{t-1}|x_t)$, which in practice cannot be done [6].
Therefore, an estimation $p_\theta(x_{t-1}|x_t)$ of $q(x_{t-1}|x_t)$ is found using a trained neural network (see 2.2.3), with θ the parameters of the neural network [6].

The reverse process is defined as follows [5]:

$$p_\theta(x_0, \ldots, x_T) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{2.11}$$

where

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

and

$$p(x_T) = \mathcal{N}_D(0, I) \qquad \text{(isotropic Gaussian distribution)}$$

Equation 2.11 gives us the full reverse process, starting from the final image x_T (sampled from noise) to the original image x_0.
Equation 2.12 gives us the reverse process at step t, where we estimate the noise that was added to the image at step t of the forward process in order to find the image x_{t-1}.
Two parameters must be estimated to find the reverse process at step t (from x_t to x_{t-1}): the mean $\mu_\theta(x_t, t)$ and the variance $\Sigma_\theta(x_t, t)$.
At first, Ho et al. [5] used neural networks to estimate both the mean μ_θ and the variance Σ_θ, but explain that estimating the variance was not necessary as there was little difference between the estimated variance $\sigma_t^2 = \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}$ and the actual values $\sigma_t^2 = \beta_t$. Thus they set $\Sigma_\theta(x_t, t) = \sigma_t^2 I = \beta_t I$.
As for the mean, it is estimated using a neural network trained to optimise a variational lower bound (VLB) of the negative log likelihood $\mathbb{E}[-\log p_\theta(x_0)]$.

2.2.1 Variational Lower Bound

Computing the negative log likelihood $\mathbb{E}[-\log p_\theta(x_0)]$ requires knowing $p_\theta(x_0)$ which means going through T steps, which is computationally expensive.
Thus, Ho et al. [5] use a variational lower bound (VLB or more commonly known as ELBO) to estimate the negative log likelihood [4].

The variational lower bound given by Dederik et al. [8] is defined as:

$$\text{VLB} = \mathbb{E}_{q_\phi(x|z)}[-\log q_\phi(z|x) + \log p_\theta(x, z)] \tag{2.13}$$

$$= -D_{KL}(q_\phi(z|x) \| p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] \tag{2.14}$$

where X, Z are jointly distributed random variables with distribution $p_\theta(x, z)$, $x \sim p_\theta$ and q_ϕ is any distribution.

This gives us:

$$\text{VLB} = \log p_\theta(x_0) - D_{KL}(q(x_{1:T}|x_0) \| p_\theta(x_{1:T}|x_0)) \tag{2.15}$$

Since we need to minimise the loss with the negative log likelihood, we write:

$$-\text{VLB} = -\log p_\theta(x_0) + D_{KL}(q(x_{1:T}|x_0) \| p_\theta(x_{1:T}|x_0)) \tag{2.16}$$

We know that the Kullback-Leibler divergence D_{KL} is non-negative, which means that we can write:

$$-\log p_\theta(x_0) \leqslant -\log p_\theta(x_0) + D_{KL}(q(x_{1:T}|x_0) \| p_\theta(x_{1:T}|x_0)) \tag{2.17}$$

$$= -\log p_\theta(x_0) + \mathbb{E}_q\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T}|x_0)}\right] \tag{2.18}$$

$$= -\log p_\theta(x_0) + \mathbb{E}_q\left[\log \frac{q(x_{1:T}|x_0)}{\frac{p_\theta(x_0, x_{1:T})}{p_\theta(x_0)}}\right] \tag{2.19}$$

$$= -\log p_\theta(x_0) + \mathbb{E}_q\left[\log \frac{q(x_{1:T}|x_0)}{\frac{p_\theta(x_{0:T})}{p_\theta(x_0)}}\right] \tag{2.20}$$

$$= -\log p_\theta(x_0) + \mathbb{E}_q\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} + \log p_\theta(x_0)\right] \tag{2.21}$$

$$= \mathbb{E}_q \left[\log \frac{q\left(x_{1:T}|x_0\right)}{p_\theta\left(x_{0:T}\right)} \right] \tag{2.22}$$

$$\text{so we obtain that } -\log p_\theta\left(x_0\right) \leqslant \mathbb{E}_q \left[\log \frac{q\left(x_{1:T}|x_0\right)}{p_\theta\left(x_{0:T}\right)} \right] \tag{2.23}$$

We can subsequently define L_{VLB} as the loss to be minimised:

$$L_{VLB} = \mathbb{E}_q \left[\log \frac{q\left(x_{1:T}|x_0\right)}{p_\theta\left(x_{0:T}\right)} \right] \tag{2.24}$$

Ho et al. [5] reformulate the loss in order to reduce variance:

$$L_{VLB} = \mathbb{E}_q \left[\log \frac{q\left(x_{1:T}|x_0\right)}{p_\theta\left(x_{0:T}\right)} \right] \tag{2.25}$$

$$= \mathbb{E}_q \left[\log \frac{\prod_{t\geqslant 1} q\left(x_t|x_{t-1}\right)}{p\left(x_T\right) \prod_{t\geqslant 1} p_\theta\left(x_{t-1}|x_t\right)} \right] \tag{2.26}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \log \frac{\prod_{t\geqslant 1} q\left(x_t|x_{t-1}\right)}{\prod_{t\geqslant 1} p_\theta\left(x_{t-1}|x_t\right)} \right] \tag{2.27}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \log \prod_{t\geqslant 1} q\left(x_t|x_{t-1}\right) - \log \prod_{t\geqslant 1} p_\theta\left(x_{t-1}|x_t\right) \right] \tag{2.28}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \sum_{t\geqslant 1} \log q\left(x_t|x_{t-1}\right) - \sum_{t\geqslant 1} \log p_\theta\left(x_{t-1}|x_t\right) \right] \tag{2.29}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \sum_{t\geqslant 1} \log q\left(x_t|x_{t-1}\right) - \log p_\theta\left(x_{t-1}|x_t\right) \right] \tag{2.30}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \sum_{t\geqslant 1} \log \frac{q\left(x_t|x_{t-1}\right)}{p_\theta\left(x_{t-1}|x_t\right)} \right] \tag{2.31}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \log \frac{q\left(x_1|x_0\right)}{p_\theta\left(x_0|x_1\right)} + \sum_{t\geqslant 2} \log \frac{q\left(x_t|x_{t-1}\right)}{p_\theta\left(x_{t-1}|x_t\right)} \right] \tag{2.32}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \log \frac{q\left(x_1|x_0\right)}{p_\theta\left(x_0|x_1\right)} + \sum_{t\geqslant 2} \log \frac{q\left(x_{t-1}|x_t, x_0\right) q\left(x_t|x_0\right)}{q\left(x_{t-1}|x_0\right) p_\theta\left(x_{t-1}|x_t\right)} \right] \tag{2.33}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \log \frac{q\left(x_1|x_0\right)}{p_\theta\left(x_0|x_1\right)} + \sum_{t\geqslant 2} \log \frac{q\left(x_{t-1}|x_t, x_0\right)}{p_\theta\left(x_{t-1}|x_t\right)} + \sum_{t\geqslant 2} \log \frac{q\left(x_t|x_0\right)}{q\left(x_{t-1}|x_0\right)} \right] \tag{2.34}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \log \frac{q\left(x_1|x_0\right)}{p_\theta\left(x_0|x_1\right)} + \log \frac{q\left(x_T|x_0\right)}{q\left(x_1|x_0\right)} + \sum_{t\geqslant 2} \log \frac{q\left(x_{t-1}|x_t, x_0\right)}{p_\theta\left(x_{t-1}|x_t\right)} \right] \tag{2.35}$$

$$= \mathbb{E}_q \left[-\log\left(p\left(x_T\right)\right) + \log \frac{q\left(x_T|x_0\right)}{p_\theta\left(x_0|x_1\right)} + \sum_{t\geqslant 2} \log \frac{q\left(x_{t-1}|x_t, x_0\right)}{p_\theta\left(x_{t-1}|x_t\right)} \right] \tag{2.36}$$

$$= \mathbb{E}_q \left[-\log\left(p_\theta\left(x_0|x_1\right)\right) + \log \frac{q\left(x_T|x_0\right)}{p\left(x_T\right)} + \sum_{t\geqslant 2} \log \frac{q\left(x_{t-1}|x_t, x_0\right)}{p_\theta\left(x_{t-1}|x_t\right)} \right] \tag{2.37}$$

$$= \mathbb{E}_q \left[D_{KL}\left(q\left(x_T|x_0\right) \| \, p\left(x_T\right)\right) \right. \tag{2.38}$$

$$+\sum_{t\geqslant 2} D_{KL}\left(q\left(x_{t-1}|x_t,x_0\right)\parallel p_\theta\left(x_{t-1}|x_t\right)\right)-\log\left(p_\theta\left(x_0|x_1\right)\right)\Bigg]$$

This formulation of the loss can be split up into 3 parts:

$$L_T = D_{KL}\left(q\left(x_T|x_0\right)\parallel p\left(x_T\right)\right) \tag{2.39}$$

$$L_{1:T-1} = \sum_{t\geqslant 2} D_{KL}\left(q\left(x_{t-1}|x_t,x_0\right)\parallel p_\theta\left(x_{t-1}|x_t\right)\right) \tag{2.40}$$

$$L_0 = -\log\left(p_\theta\left(x_0|x_1\right)\right) \tag{2.41}$$

$$\text{so that} \qquad L_{VLB} = L_T + L_{1:T-1} + L_0 \tag{2.42}$$

It is worth noting that at equation (2.33), the terms of q are conditioned on x_0; this is in order to be able to easily compute the forward process posteriors, as they are much easier to find given the original image x_0. Ho et al. [5] define it as:

$$q\left(x_{t-1}|x_t,x_0\right) = \mathcal{N}\left(x_{t-1};\tilde{\mu}_t\left(x_t,x_0\right),\tilde{\beta}_t I\right) \tag{2.43}$$

Using Bayes' theorem, we find:

$$q\left(x_{t-1}|x_t,x_0\right) = q\left(x_t|x_{t-1},x_0\right)\frac{q\left(x_{t-1}|x_0\right)}{q\left(x_t|x_0\right)} \tag{2.44}$$

Assuming that $q\left(x_t|x_{t-1},x_0\right) = \mathcal{N}\left(x_{t-1};\tilde{\mu}_t\left(x_t,x_0\right),\tilde{\beta}_t I\right)$, then:

$$q\left(x_{t-1}|x_t,x_0\right) = \mathcal{N}\left(x_t;\sqrt{1-\beta_t}x_{t-1},\beta_t I\right)\frac{\mathcal{N}\left(\sqrt{\bar{\alpha}_{t-1}}x_0,\left(1-\bar{\alpha}_{t-1}\right)I\right)}{\mathcal{N}\left(\sqrt{\bar{\alpha}_t}x_0,\left(1-\bar{\alpha}_t\right)I\right)} \tag{2.45}$$

$$= \frac{1}{\sqrt{2\pi\left(1-\bar{\alpha}_t\right)\left(1-\bar{\alpha}_{t-1}\right)\left(\sqrt{1-\beta_t}\right)}}\, e^{-\frac{1}{2}\left(\frac{\left(x_t-\sqrt{1-\beta_t}x_{t-1}\right)^2}{\beta_t}+\frac{\left(x_{t-1}-\sqrt{\bar{\alpha}_{t-1}}x_0\right)^2}{1-\bar{\alpha}_{t-1}}-\frac{\left(x_t-\sqrt{\bar{\alpha}_t}x_0\right)^2}{1-\bar{\alpha}_t}\right)} \tag{2.46}$$

Note: we rewrote $q\left(x_t|x_{t-1},x_0\right)$ as $q\left(x_t|x_{t-1}\right)$ (2.3) in equation (2.45) since x_0 adds no new information not already present in x_{t-1}.

We can ignore the constant factor in front of (2.46) and thus obtain:

$$\propto \exp\left(-\frac{1}{2}\left(\frac{x_t^2-2\sqrt{\alpha_t}x_{t-1}x_t+\alpha_t x_{t-1}^2}{\beta_t}+\frac{x_{t-1}^2-2\sqrt{\bar{\alpha}_{t-1}}x_0x_{t-1}+\bar{\alpha}_{t-1}x_0^2}{1-\bar{\alpha}_{t-1}}-\frac{\left(x_t-\sqrt{\alpha_t}x_0\right)^2}{1-\bar{\alpha}_t}\right)\right) \tag{2.47}$$

$$= \exp\left(-\frac{1}{2}\left(\frac{x_t^2}{\beta_t}-\frac{2\sqrt{\alpha_t}x_tx_{t-1}}{\beta_t}+\frac{\alpha_t x_{t-1}^2}{\beta_t}+\frac{x_{t-1}^2}{1-\bar{\alpha}_{t-1}}-\frac{2\sqrt{\bar{\alpha}_{t-1}}x_0x_{t-1}}{1-\bar{\alpha}_{t-1}}+\frac{\bar{\alpha}_{t-1}x_0^2}{1-\bar{\alpha}_{t-1}}-\frac{\left(x_t-\sqrt{\alpha_t}x_0\right)^2}{1-\bar{\alpha}_t}\right)\right) \tag{2.48}$$

$$= \exp\left(-\frac{1}{2}\left(x_{t-1}^2\left(\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}}\right)+x_{t-1}\left(\frac{-2\sqrt{\alpha_t}x_t}{\beta_t}-\frac{2\sqrt{\bar{\alpha}_{t-1}}x_0}{1-\bar{\alpha}_{t-1}}\right)+C\left(x_t,x_0\right)\right)\right) \tag{2.49}$$

where:

$$C\left(x_t,x_0\right) = \frac{x_t^2}{\beta_t}+\frac{\bar{\alpha}_{t-1}x_0^2}{1-\bar{\alpha}_{t-1}}-\frac{\left(x_t-\sqrt{\alpha_t}x_0\right)^2}{1-\bar{\alpha}_t} \tag{2.50}$$

Considering $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=0}^{t} \alpha_i$, we can simplify the expression to:

$$= \exp\left(-\frac{1}{2}\left(x_{t-1}^2\left(\frac{\alpha_t}{1-\alpha_t} + \frac{1}{1-\bar{\alpha}_{t-1}}\right) - 2\left(\frac{\sqrt{\alpha_t}x_t}{1-\alpha_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}x_0}{1-\bar{\alpha}_{t-1}}\right)x_{t-1} + C\left(x_t, x_0\right)\right)\right) \tag{2.51}$$

$$= \exp\left(-\frac{1}{2}\left(x_{t-1}^2\left(\frac{\alpha_t\left(1-\bar{\alpha}_{t-1}\right) + \left(1-\alpha_t\right)}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)}\right) - 2\left(\frac{\sqrt{\alpha_t}x_t}{1-\alpha_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}x_0}{1-\bar{\alpha}_{t-1}}\right)x_{t-1} + C\left(x_t, x_0\right)\right)\right) \tag{2.52}$$

$$= \exp\left(-\frac{1}{2}\left(x_{t-1}^2\left(\frac{\alpha_t - \alpha_t\bar{\alpha}_{t-1} + 1 - \alpha_t}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)}\right) - 2\left(\frac{\sqrt{\alpha_t}x_t}{1-\alpha_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}x_0}{1-\bar{\alpha}_{t-1}}\right)x_{t-1} + C\left(x_t, x_0\right)\right)\right) \tag{2.53}$$

$$= \exp\left(-\frac{1}{2}\left(x_{t-1}^2\left(\frac{1-\bar{\alpha}_{t-1}}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)}\right) - 2\left(\frac{\sqrt{\alpha_t}x_t}{1-\alpha_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}x_0}{1-\bar{\alpha}_{t-1}}\right)x_{t-1} + C\left(x_t, x_0\right)\right)\right) \tag{2.54}$$

$$= \exp\left(-\frac{1}{2}\left(\left(\frac{1-\bar{\alpha}_{t-1}}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)}\right)\left(x_{t-1}^2 - 2\frac{\frac{\sqrt{\alpha_t}x_t}{1-\alpha_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}x_0}{1-\bar{\alpha}_{t-1}}}{\frac{1-\bar{\alpha}_{t-1}}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)}}x_{t-1} + \frac{C\left(x_t, x_0\right)}{\frac{1-\bar{\alpha}_{t-1}}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)}}\right)\right)\right) \tag{2.55}$$

$$= \exp\left(-\frac{1}{2}\left(\left(\frac{1-\bar{\alpha}_{t-1}}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)}\right)\left(x_{t-1}^2 - 2\frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)x_t + \sqrt{\bar{\alpha}_{t-1}}\left(1-\alpha_t\right)x_0}{1-\bar{\alpha}_t} + \frac{C\left(x_t, x_0\right)}{\frac{1-\bar{\alpha}_{t-1}}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)}}\right)\right)\right) \tag{2.56}$$

As for the final term, since:

$$C\left(x_t, x_0\right) = \frac{x_t^2}{\beta_t} + \frac{\bar{\alpha}_{t-1}x_0^2}{1-\bar{\alpha}_{t-1}} - \frac{\left(x_t - \sqrt{\alpha}_t x_0\right)^2}{1-\bar{\alpha}_t} \tag{2.57}$$

$$= \frac{x_t^2}{1-\alpha_t} + \frac{\bar{\alpha}_{t-1}x_0^2}{1-\bar{\alpha}_{t-1}} - \frac{x_t^2 - 2\sqrt{\alpha}_t x_t x_0 + \alpha_t x_0^2}{1-\bar{\alpha}_t} \tag{2.58}$$

We obtain:

$$C\left(x_t, x_0\right) / \frac{1-\bar{\alpha}_{t-1}}{\left(1-\alpha_t\right)\left(1-\bar{\alpha}_{t-1}\right)} \tag{2.59}$$

$$= \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_{t-1}} \left(\frac{x_t^2}{1-\alpha_t} + \frac{\bar{\alpha}_{t-1} x_0^2}{1-\bar{\alpha}_{t-1}} - \frac{x_t^2 - 2\sqrt{\alpha_t} x_t x_0 + \alpha_t x_0^2}{1-\bar{\alpha}_t} \right) \tag{2.60}$$

$$= x_t^2 \left(\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} - \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)^2} \right) \tag{2.61}$$
$$+ x_0^2 \left(\frac{(1-\alpha_t)(\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} - \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})\bar{\alpha}_t}{(1-\bar{\alpha}_t)^2} \right)$$
$$+ \frac{2x_t x_0 \sqrt{\bar{\alpha}_t}(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)^2}$$

$$= x_t^2 \left(\frac{(1-\bar{\alpha}_t)(1-\bar{\alpha}_{t-1}) - (1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)^2} \right) \tag{2.62}$$
$$+ x_0^2 \left(\frac{(1-\bar{\alpha}_t)(1-\alpha_t)\bar{\alpha}_{t-1} - (1-\alpha_t)(1-\bar{\alpha}_{t-1})\bar{\alpha}_t}{(1-\bar{\alpha}_t)^2} \right)$$
$$+ \frac{2x_t x_0 \sqrt{\bar{\alpha}_t}(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)^2}$$

$$= x_t^2 \frac{(1-\bar{\alpha}_{t-1})^2 \alpha_t}{(1-\bar{\alpha}_t)^2} + x_0^2 \frac{(1-\bar{\alpha}_t)^2 \bar{\alpha}_{t-1}}{(1-\bar{\alpha}_t)^2} + \frac{2x_t x_0 \sqrt{\bar{\alpha}_t}(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)^2} \tag{2.63}$$

$$= \frac{\left(x_t \sqrt{\alpha_t}(1-\bar{\alpha}_{t-1}) + x_0 \sqrt{\bar{\alpha}_{t-1}}(1-\bar{\alpha}_t)\right)^2}{(1-\bar{\alpha}_t)^2} \tag{2.64}$$

$$= \left(\frac{\left(x_t \sqrt{\alpha_t}(1-\bar{\alpha}_{t-1}) + x_0 \sqrt{\bar{\alpha}_{t-1}}(1-\bar{\alpha}_t)\right)}{(1-\bar{\alpha}_t)} \right)^2 \tag{2.65}$$

We can now replace this value into the original expression:

$$q(x_{t-1}|x_t, x_0) \propto \exp\left(-\frac{1}{2} \left(\left(\frac{1}{\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_{t-1}}} \right) \right. \right. \tag{2.66}$$
$$\left(x_{t-1}^2 - 2 \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1}) x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t) x_0}{1-\bar{\alpha}_t} \right.$$
$$\left. \left. \left. + \left(\frac{\left(x_t \sqrt{\alpha_t}(1-\bar{\alpha}_{t-1}) + x_0 \sqrt{\bar{\alpha}_{t-1}}(1-\bar{\alpha}_t)\right)}{(1-\bar{\alpha}_t)} \right)^2 \right) \right) \right) \tag{2.67}$$

$$= \exp\left(-\frac{1}{2} \left(\frac{1}{\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_{t-1}}} \right) \right. \tag{2.68}$$
$$\left. \left(x_{t-1} - \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1}) x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t) x_0}{1-\bar{\alpha}_t} \right)^2 \right)$$

which through normalisation gives:

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t}, \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}I\right) \tag{2.69}$$

$$= \mathcal{N}\left(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I\right) \tag{2.70}$$

We have therefore obtained $\tilde{\mu}(x_t, x_0)$ and $\tilde{\beta}_t$ mentioned in Ho et al.'s [5] paper.
The authors also rearrange $\tilde{\mu}(x_t, x_0)$ in order to remove the dependency on x_0:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \tag{2.71}$$

$$\Leftrightarrow x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon}{\sqrt{\bar{\alpha}_t}} \tag{2.72}$$

$$\Rightarrow \tilde{\mu}(x_t, t) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon}{\sqrt{\bar{\alpha}_t}}}{1-\bar{\alpha}_t} \tag{2.73}$$

$$= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + (1-\alpha_t)\left(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon\right)\frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}}}{1-\bar{\alpha}_t} \tag{2.74}$$

$$= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t}{1-\bar{\alpha}_t} + \frac{(1-\alpha_t)\left(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon\right)\frac{1}{\sqrt{\alpha_t}}}{1-\bar{\alpha}_t} \tag{2.75}$$

$$= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{(1-\alpha_t)}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}x_t - \frac{(1-\alpha_t)(\sqrt{1-\bar{\alpha}_t})}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}\epsilon \tag{2.76}$$

$$= \left(\frac{\sqrt{\alpha_t}\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)} + \frac{(1-\alpha_t)}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}\right)x_t - \frac{(1-\alpha_t)(\sqrt{1-\bar{\alpha}_t})}{\sqrt{(1-\bar{\alpha}_t)}\sqrt{(1-\bar{\alpha}_t)}\sqrt{\alpha_t}}\epsilon \tag{2.77}$$

$$= \left(\frac{\alpha_t(1-\bar{\alpha}_{t-1}) + (1-\alpha_t)}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}\right)x_t - \frac{(1-\alpha_t)}{\sqrt{(1-\bar{\alpha}_t)}\sqrt{\alpha_t}}\epsilon \tag{2.78}$$

$$= \left(\frac{\alpha_t - \bar{\alpha}_t + 1 - \alpha_t}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}\right)x_t - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)}\sqrt{\alpha_t}}\epsilon \tag{2.79}$$

$$= \frac{1}{\sqrt{\alpha_t}}x_t - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)}\sqrt{\alpha_t}}\epsilon \tag{2.80}$$

$$= \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon\right) \tag{2.81}$$

Another important point is that the term L_T (2.39) can be omitted since q has no learnable parameters and together with the fact that it will almost become a Gaussian distribution, the Kullback-Leibler Divergence $D_{KL}\left(q\left(x_T|x_0\right) \| p\left(x_T\right)\right)$ between a nearly Gaussian distribution and a Gaussian distribution $p\left(x_T\right) = \mathcal{N}\left(x_t; 0, I\right)$ will be close to 0 [5].
We thus find ourselves with:

$$L_{VLB} = \mathbb{E}_q\left[\sum_{t\geqslant 2} D_{KL}\left(q\left(x_{t-1}|x_t, x_0\right) \| p_\theta\left(x_{t-1}|x_t\right)\right) - \log\left(p_\theta\left(x_0|x_1\right)\right)\right] \tag{2.82}$$

2.2.2 Simplified training objective

Ho et al. [5] simplify the training objective by making several changes in order to obtain an easy quantity for minimising the loss.

First of all, $L_{1:T-1}$ can be written as such:

$$L_{1:T-1} = D_{KL}\left(q\left(x_{t-1}|x_t, x_0\right) \| p_\theta\left(x_{t-1}|x_t\right)\right) \tag{2.83}$$

$$= D_{KL}\left(\mathcal{N}(x_{t-1}; \tilde{\mu}_t\left(x_t, t\right), \sigma_t^2 I) \| \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I\right)\right) \tag{2.84}$$

The Kullback-Leibler divergence between the two Gaussian distributions of dimension D is given by:

$$D_{KL}\left(\mathcal{N}(x_{t-1};\tilde{\mu}_t(x_t,t),\sigma_t^2 I)\|\,\mathcal{N}\left(x_{t-1};\mu_\theta(x_t,t),\sigma_t^2 I\right)\right) \tag{2.85}$$

$$= \frac{1}{2}\left(\mathrm{tr}\left((\sigma_t^2 I)^{-1}(\sigma_t^2 I)\right) - D + (\mu_\theta-\tilde{\mu}_t)^T(\sigma_t^2 I)^{-1}(\mu_\theta-\tilde{\mu}_t) + \log\frac{\det(\sigma_t^2 I)}{\det(\sigma_t^2 I)}\right) \tag{2.86}$$

$$= \frac{1}{2}\left(D - D + (\mu_\theta-\tilde{\mu}_t)^T(\sigma_t^2 I)^{-1}(\mu_\theta-\tilde{\mu}_t) + \log 1\right) \tag{2.87}$$

$$= \frac{1}{2}\left((\mu_\theta-\tilde{\mu}_t)^T(\sigma_t^2 I)^{-1}(\mu_\theta-\tilde{\mu}_t)\right) \tag{2.88}$$

$$= \frac{1}{2\sigma_t^2}\left((\mu_\theta-\tilde{\mu}_t)^T(\mu_\theta-\tilde{\mu}_t)\right) \tag{2.89}$$

$$= \frac{1}{2\sigma_t^2}\|\mu_\theta-\tilde{\mu}_t\|_2^2 \tag{2.90}$$

However, Ho et al. [5] rewrite this formula so that it depends only on the noise ϵ:

$$\frac{1}{2\sigma_t^2}\|\mu_\theta-\tilde{\mu}_t\|_2^2 = \frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon\right) - \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon_\theta\right)\right\|_2^2 \tag{2.91}$$

$$= \frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\alpha_t}}\frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)}}(\epsilon-\epsilon_\theta)\right\|_2^2 \tag{2.92}$$

$$= \frac{1}{2\sigma_t^2}\frac{1}{\alpha_t}\frac{(1-\alpha_t)^2}{1-\bar{\alpha}_t}\|\epsilon-\epsilon_\theta\|_2^2 \tag{2.93}$$

$$= \frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\epsilon-\epsilon_\theta\|_2^2 \tag{2.94}$$

Furthermore, they found that simply ignoring the factor in front gave better results, giving us:

$$L_{1:T-1} = \|\epsilon-\epsilon_\theta\|_2^2 \tag{2.95}$$

As for the second term L_0, the authors define $p_\theta(x_0|x_1)$ as:

$$p_\theta(x_0|x_1) = \prod_{i=1}^{D}\int_{\delta_-(x_0^i)}^{\delta_+(x_0^i)}\mathcal{N}(x;\mu_\theta^i(x_1,1),\sigma_1^2)\,dx \tag{2.96}$$

where $\delta_+(x)$ and $\delta_-(x)$ are defined as:

$$\delta_+(x) = \begin{cases}\infty & \text{if } x=1\\ x+\frac{1}{255} & \text{if } x<1\end{cases} \tag{2.97}$$

$$\delta_-(x) = \begin{cases}-\infty & \text{if } x=-1\\ x-\frac{1}{255} & \text{if } x>-1\end{cases} \tag{2.98}$$

with D the data dimensionality and i one coordinate from the image.

This is a decoder for the last step in order to get pixel values between $\{0,1,\ldots,255\}$ from the values in $[-1,1]$ used for the neural network [5, 6]
However the authors also ignore the entire term, leaving us with:

$$L_{\text{simple}}(\theta) = \mathbb{E}_q\left[\|\epsilon-\epsilon_\theta\|_2^2\right] \tag{2.99}$$

2.2.3 Model architecture

Ho et al. [5] as well as the following papers [7, 6, 9] use a U-Net style architecture for the neural network to estimate the noise.

The U-Net architecture is a convolutional neural network (CNN) that was originally designed for image segmentation in biomedical imaging [10]. It consists of an encoder (or contracting path) and a decoder (or

expansive path) that are connected by a bottleneck.
The encoder takes in the noisy image as input and repeatedly applies unpadded convolutions followed by max pooling layers in order to reduce the spatial dimensions of the image [10].
The bottleneck contains 3 unpadded convolutions each followed by a rectified linear unit (ReLU) activation function [10].
The decoder then applies up-convolutions followed by unpadded convolutions to increase the spatial dimensions of the image, with at each step a concatenation of the feature maps from the encoder [10].

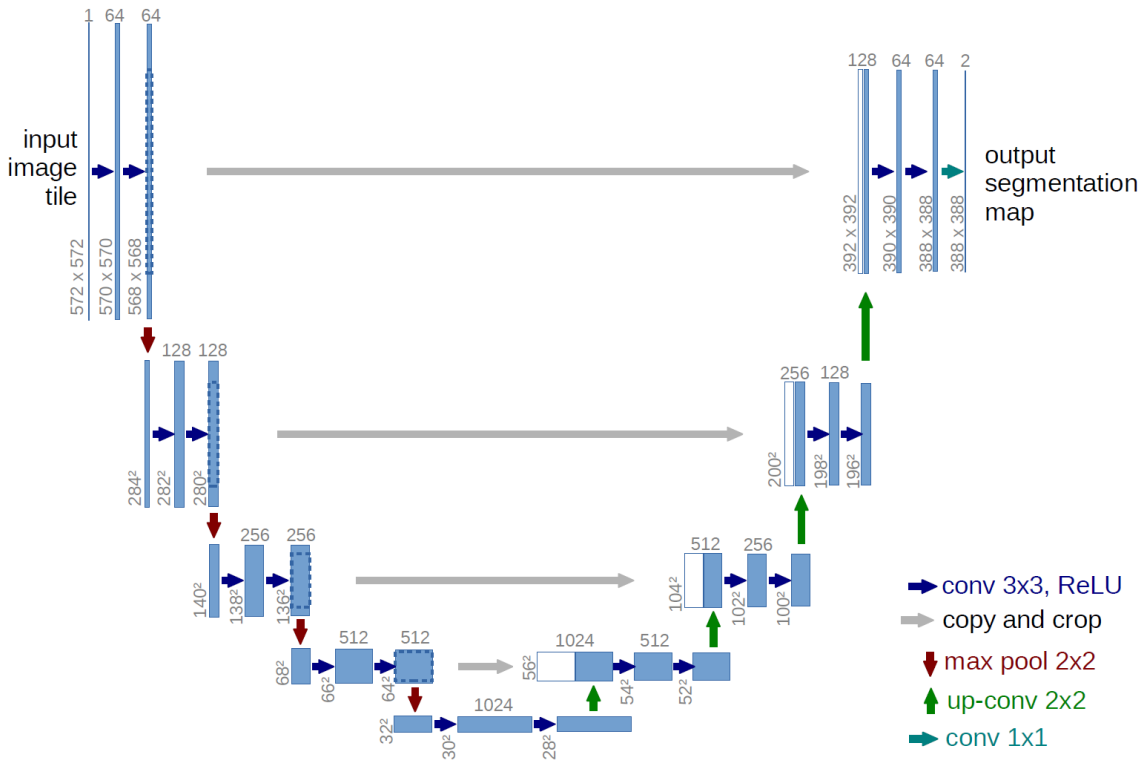

Figure 2.1: Original U-Net architecture [10]

However, Ho et al. [5] made some changes to the original U-Net architecture.
First, they state that they used 4 feature map resolutions for their 32×32 resolution model, and 6 for the 256×256 resolution model. For each of these feature maps, they used two convolutional residual blocks similar of those of a Wide ResNet [11] consisting of a convolutional layer, a ReLU activation function, a convolutional layer, a group normalisation layer, and another ReLU activation function.
The "residual" part of the block comes from the fact that the result at the end of the block on the encoder side is concatenated with the result at the end of the block on the decoder side in order to consider features that could have been lost during the downsampling [12].
They used group normalisation [13] rather than weight normalisation [14], which is a normalisation technique that divides the channels into groups and computes the mean and standard deviation for each group [5].
Finally, they state that they used self-attention layers from the Transformer architecture [15] at the 16×16 resolution between the two convolutional residual blocks [5].

Since the model needs to be able to predict the noise at any timestep t, the authors added a time embedding to the input of the network using the sinusoidal positional embedding from the Transformer architecture [15], which allows the parameters to be shared across all timesteps.

The sinusoidal positional embedding is given by [15]:

$$\mathrm{PE}_{(\mathrm{pos},2\mathrm{i})} = \sin\left(\frac{\mathrm{pos}}{10000^{2\mathrm{i}/\mathrm{d}_{\mathrm{model}}}}\right) \tag{2.100}$$

$$\mathrm{PE}_{(\mathrm{pos},2\mathrm{i}+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{2\mathrm{i}/\mathrm{d}_{\mathrm{model}}}}\right) \tag{2.101}$$

where pos is the position and i is the dimension

Ho et al. [5] present two algorithms: one for training the model and one for sampling from the model:

Algorithm 1 Training

repeat
  $x_0 \sim q(x_0)$ ▷ Sample an image x_0 from the dataset
  $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
  $\epsilon \sim \mathcal{N}(0, I)$
  Take gradient descent step on:
    $\nabla_\theta \| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \|^2$
until converged

Algorithm 2 Sampling

$x_T \sim \mathcal{N}(0, I)$
for $t = T, \ldots, 1$ **do**
  $z \sim \mathcal{N}(0, I)$ if $t > 0$, else $z = 0$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$
end for
return x_0

3 Score-based formulation

The DDPM paper [5] allows us to predict either the next denoised image, the mean of the next denoised image or the noise itself.
However they mention another formulation of the DDPM model that is equivalent: a score-based generative modeling version.

3.1 Score matching

This formulation uses score matching [16]: either denoising score matching [17] or sliced score matching [18]. Essentially, score matching avoids optimising the likelihood which can sometimes be difficult to find because of normalisation constants and instead optimises a score function which is the gradient of the log-likelihood. Suppose we have a probability density function $p_\theta(x)$ that models an i.i.d. dataset $\{x_i\}_{i=1}^N$ of the p.d.f. $p(x)$.

Let $f_\theta(x)$ be a real valued function which is an unnormalized density function and $Z_\theta = \int e^{-f_\theta(x)} dx$ be the normalisation constant such that $\int p_\theta(x)\ dx = 1$, where $p_\theta(x)$ is defined as:

$$p_\theta(x) = \frac{e^{-f_\theta(x)}}{Z_\theta} \tag{3.1}$$

However for a lot of models, Z_θ cannot be computed [19].

The score function[1] $s_\theta : \mathbb{R}^D \to \mathbb{R}^D$ is defined as [20]:

$$s_\theta(x) = \nabla_x \log p(x) \tag{3.2}$$

If we try to train a model to optimise this score function, we get:

$$\begin{aligned} s_\theta(x) &= \nabla_x \log p_\theta(x) && (3.3)\\ &= \nabla_x \log \frac{e^{-f_\theta(x)}}{Z_\theta} && (3.4)\\ &= \nabla_x \left(-f_\theta(x) - \log Z_\theta\right) && (3.5)\\ &= -\nabla_x f_\theta(x) - \nabla_x \log Z_\theta && (3.6)\\ &= -\nabla_x f_\theta(x) && (3.7) \end{aligned}$$

Since Z_θ is a constant with respect to x, $\nabla_x \log Z_\theta = 0$ which means we don't need to compute the intractable normalisation constant Z_θ.
This means we could train the score network s_θ to match the true score function $s(x) = \nabla_x \log p(x)$.
The model should then optimise the following Fisher divergence $J(\theta)$ [16, 19]:

$$J(\theta) = \frac{1}{2}\mathbb{E}_{x\sim p(x)}\left[\|s_\theta(x) - s(x)\|_2^2\right] \tag{3.8}$$

However, since we don't know the true score function $s(x)$ which is the score function of the true data distribution $p(x)$, another formulation was shown by Hyvärinen [16] to be equivalent:

$$J(\theta) = \mathbb{E}_{x\sim p(x)}\left[\operatorname{tr}(\nabla_x s_\theta(x)) + \frac{1}{2}\|s_\theta(x)\|_2^2\right] \tag{3.9}$$

This is the score matching objective we want to minimise [16].
It can also be computed using the datapoints with the following formula [16]:

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \operatorname{tr}(\nabla_x s_\theta(x_i)) + \frac{1}{2}\|s_\theta(x_i)\|_2^2 \tag{3.10}$$

A problem that arises with this formulation is that the Jacobian matrix $\nabla_x s_\theta(x)$ is not suitable for deep neural networks as it requires backpropagation to compute all its diagonal elements to find the trace [20, 18]. To solve this problem, Song et al. [20] propose either denoising score matching or sliced score matching.

[1] In ML, the score function used is the Stein score, which is the data gradient of the log likelihood. However in statistics, the Fisher score is used and is the parameter gradient of the log likelihood $s_F(x) = \nabla_\theta \log p_\theta(x)$

Denoising score matching [17] is a method that slightly noises a datapoint and then score matches the denoised datapoint. This is done by convolving datapoints sampled from the true data distribution p with a noising kernel q_σ such that $p(x) \approx q_\sigma(x)$. Given a datapoint x, we get a noised datapoint $\tilde{x}$ by sampling from $q_\sigma(\tilde{x}|x)$.
The score matching now estimates the score of the noised data distribution $q_\sigma(\tilde{x})$, which leads us to the following objective [20, 17]:

$$J(\theta) = \frac{1}{2}\mathbb{E}_{q_\sigma(\tilde{x}|x)p(x)}\left[\|s_\theta(\tilde{x}) - \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)\|_2^2\right] \tag{3.11}$$

Sliced score matching [18] projects the vectors of the vector field $s_\theta(x)$ onto random directions to obtain a scalar field. This will approximate the trace of the Jacobian matrix and it becomes much easier to compute [20].
We get the following objective [18]:

$$J(\theta) = \mathbb{E}_{p_v}\mathbb{E}_{x\sim p(x)}\left[v^T \nabla_x s_\theta(x)\, v + \frac{1}{2}\|s_\theta(x)\|_2^2\right] \tag{3.12}$$

The approximation of the Jacobian matrix $v^T \nabla_x s_\theta(x)\, v$ can be rewritten as [18]:

$$v^T \nabla_x s_\theta(x)\, v = v^T \nabla_x (v^T s_\theta(x)) \tag{3.13}$$

The gradient of the inner product $\nabla_x(v^T s_\theta(x))$ requires only one backpropagation pass and requires another inner product with v^T to approximate the trace [18].

3.2 Sampling

Using the score function mentioned above, we can sample from the model by using Langevin dynamics [20, 21], which will give us the definition of score-based generative modeling [20].

Let $x_0 \sim \pi(x)$ where $\pi(x)$ is an arbitrary distribution (such as a Gaussian distribution), $\epsilon > 0$ a fixed step size and $z_t \sim \mathcal{N}_D(0, I)$.
Using Stochastic Gradient Langevin Dynamics (SGLD) [20, 21], we can recursively sample using the following formula:

$$x_t = x_{t-1} + \frac{\epsilon}{2}\nabla_x \log p(x_{t-1}) + \sqrt{\epsilon} z_t \tag{3.14}$$

When $T \to \infty$ and $\epsilon \to 0$, the distribution of x_T will converge to the true data distribution $p(x)$ therefore the sample will become one from the true data distribution [20, 21].
z_t is used to add some Gaussian noise at each step because if there was no stochasticity at each step, the sampling would become deterministic and all the samples would essentially be the same.
Since we don't know the true data distribution $p(x)$, we can use the score function $s_\theta(x)$ trained to match the true score function $s(x)$ to sample from the model.

$$x_t = x_{t-1} + \frac{\epsilon}{2} s_\theta(x_{t-1}) + \sqrt{\epsilon} z_t \tag{3.15}$$

There is however a problem with this method: the score function is well approximated at high density regions where there are a lot of data points but not at low density regions [20].
Song and Ermon [20] show this in figure 3.1 by comparing the real score function $\nabla_x \log p(x)$ and the estimated score function $s_\theta(x)$ for a mixture of two Gaussian distributions.

This means that when sampling using Langevin dynamics, the samples will ultimately be incorrect [20]. To correct this, Song and Ermon [20] use annealed Langevin dynamics.
The idea is to add noise to the data points in order to improve the score function estimation in low density regions [22]. However adding too much noise will make the score function estimation worse in high density regions [22].
Therefore, they control the amount of noise added using a scale of different standard deviations σ_i with $i = 1, 2, \ldots, L$ such that $\sigma_1 < \sigma_2 < \cdots < \sigma_L$ [22, 20].
To sample, the same Langevin dynamics method is used going from $i = L, L-1, \ldots, 1$ with σ_i decreasing at each step. This is the annealed Langevin dynamics method they used, and the results can be seen in figure 3.2, where we clearly see that the annealed Langevin dynamics samples are much more similar to the original data samples than the simple Langevin dynamics samples [22, 20].

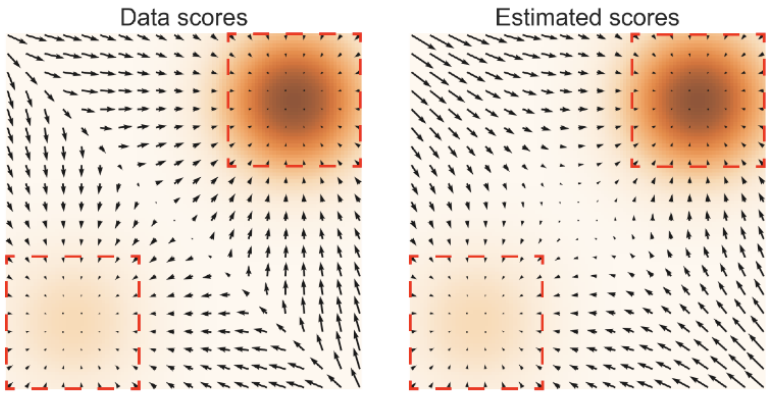

Figure 3.1: Data scores represent the real score function $\nabla_x \log p(x)$ and estimated scores reperesent the estimated score function $s_\theta(x)$. We can see that the regions in the red rectangles are the darker high density regions where the score function is well approximated, while the lighter lower density regions outside the red rectangles were not well estimated. Source: [20]

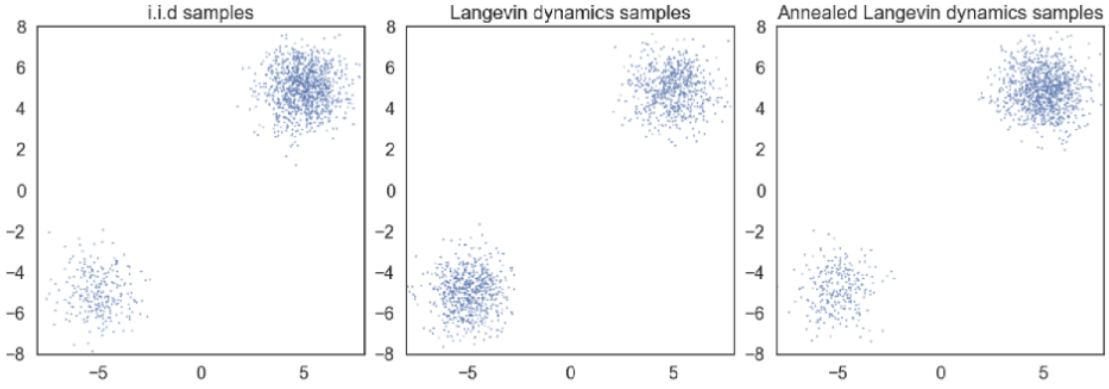

Figure 3.2: Left: Samples from data distribution. Middle: Samples from model using Langevin dynamics. Right: Samples from model using annealed Langevin dynamics. Source: [20]

3.3 Noise Conditional Score Networks

Noise Conditional Score Networks (NCSN) are a score-based generative model that use a neural network to estimate the score function of the previously mentioned noised distribution and then use annealed Langevin dynamics to sample from the model [20].

Given the increasing sequence of standard deviations σ_i with $i = 1, 2, \ldots, L$, the perturbed data distribution is defined as [20]:

$$q_{\sigma_i}(x) = \int p(t)\, \mathcal{N}(x; t, \sigma_i^2)\, dt \tag{3.16}$$

Song and Ermon [20] then define the Noise Conditional Score Network s_θ that estimates the scores of all the perturbed data distributions $q_{\sigma_i}(x)$ given a data point x:

$$s_\theta : \mathbb{R}^D \times \mathbb{R} \to \mathbb{R}^D \tag{3.17}$$

$$s_\theta(x, \sigma) \approx \nabla_x \log q_\sigma(x) \qquad \forall \sigma \in \{\sigma_1, \sigma_2, \ldots, \sigma_L\} \tag{3.18}$$

To train the NCSN, the authors chose denoising score matching [17] as its goal is to estimate the score function of a noised data distribution while also being quicker than sliced score matching [20, 18].

Let $q_\sigma(\tilde{x}|x) = \mathcal{N}(\tilde{x}|x, \sigma^2, I)$ be the noise distribution that noises the data point x to get the noised data point $\tilde{x}$. Its score function is given by [20]:

$$\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) = -\frac{\tilde{x} - x}{\sigma^2} \tag{3.19}$$

Let $\lambda(\sigma_i)$ be a positive coefficient function, often chosen to be $\lambda(\sigma_i) = \sigma_i^2$ [22, 20].
The denoising score matching objective is given by [20]:

$$\mathcal{L}(\theta, \{\sigma_i\}_{i=1}^L) = \frac{1}{2L} \sum_{i=1}^{L} \lambda(\sigma_i)\, \mathbb{E}_p(x) \mathbb{E}_{\tilde{x}\sim\mathcal{N}(x,\sigma^2 I)} \left[\| s_\theta(\tilde{x}, \sigma_i) + \frac{\tilde{x} - x}{\sigma_i^2} \|_2^2 \right] \tag{3.20}$$

To sample from the model, Song and Ermon [20] define an algorithm that uses the previously mentioned annealed Langevin dynamics:

Algorithm 3 Sampling from NCSN

Require: $\{\sigma_i\}_{i=1}^L$, ϵ, T
1: $\tilde{x}_0 \sim \pi(x)$
2: **for** $i = 1, 2, ..., L$ **do**
3: $\quad \alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$
4: $\quad$ **for** $t = 1, 2, ..., T$ **do**
5: $\quad\quad z_t \sim \mathcal{N}(0, I)$
6: $\quad\quad \tilde{x}_t \leftarrow \tilde{x}_{t-1} + \frac{\alpha_i}{2} s_\theta(\tilde{x}_{t-1}, \sigma_i) + \sqrt{\alpha_i} z_t$
7: $\quad$ **end for**
8: $\quad \tilde{x}_0 \leftarrow \tilde{x}_T$
9: **end for**
10: **return** $\tilde{x}_T$

3.4 Equivalence between DDPM and NCSN

Ho et al.'s DDPM paper [5] showed that the DDPM model is equivalent to the NCSN model with denoising score matching and annealed Langevin dynamics.
This can be seen by the fact that sampling algorithm of the DDPM paper (2) is in fact very similar to the sampling algorithm of the NCSN paper (3) but rather than estimating a score, the DDPM model estimates the noise [5].
Similarly, the score function $s_\theta(x, \sigma)$ can be rewritten in terms of the DDPM's forward process as [23]:

$$\begin{aligned}
s_\theta(x, \sigma) \approx \nabla_x \log q_\sigma(x) \Leftrightarrow s_\theta(x_t, t) &\approx \nabla_{x_t} \log q(x_t) \qquad \text{(see 2.3)} && (3.21)\\
&= \mathbb{E}_{q(x_0)} \left[\nabla_{x_t} \log q(x_t \,|\, x_0) \right] && (3.22)\\
&= \mathbb{E}_{q(x_0)} \left[\nabla_{x_t} \log \mathcal{N}(\sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) I) \right] && (3.23)\\
&= \mathbb{E}_{q(x_0)} \left[\nabla_{x_t} \log e^{-\frac{1}{2}\left(\frac{x_t - \sqrt{\bar{\alpha}_t} x_0}{\sqrt{1-\bar{\alpha}_t}}\right)^T \left(\frac{x_t - \sqrt{\bar{\alpha}_t} x_0}{\sqrt{1-\bar{\alpha}_t}}\right)} \right] && (3.24)\\
&= \mathbb{E}_{q(x_0)} \left[\Sigma^{-1} (x_t - \sqrt{\bar{\alpha}_t} x_0) \right] && (3.25)\\
&= \mathbb{E}_{q(x_0)} \left[\Sigma^{-1} \left(x_t - \sqrt{\bar{\alpha}_t} \left(\frac{x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon_\theta}{\sqrt{\bar{\alpha}_t}} \right) \right) \right] && (3.26)\\
&= \mathbb{E}_{q(x_0)} \left[\Sigma^{-1} \left(\sqrt{1 - \bar{\alpha}_t} \epsilon_\theta \right) \right] && (3.27)\\
&= -\frac{1}{1 - \bar{\alpha}_t} \sqrt{1 - \bar{\alpha}_t} \epsilon_\theta && (3.28)\\
&= -\frac{\epsilon_\theta}{\sqrt{1 - \bar{\alpha}_t}} && (3.29)
\end{aligned}$$

We can see that this quantity is used in sampling algorithm (2) of the DDPM paper [5].

4 Improvements upon DDPMs

As mentioned previously, the DDPM paper [5] had certain choices that limited its performance.
Several papers [6, 9] following it have improved upon the original DDPM in various ways.

4.1 Improved likelihood

Ho et al. [5] admit that their model, despite its high sample quality, did not have a competitive log-likelihood in comparison to other likelihood-based models such as VAEs and autoregressive models [6].

The *Improved Denoising Diffusion Probabilistic Models* paper published by Nichol et al. [6] in 2021 implemented certain changes in order to improve the log-likelihood of the model not only on simpler datasets such as CIFAR-10 [24] that the original DDPM was trained on, but also on more complex datasets such as ImageNet [8, 6, 25].

In order to improve the log-likelihood, Nichol et al. [6] started by investigating the fixed variance $\Sigma_\theta(x_t, t) = \sigma_t^2 I = \beta_t I$ of the original DDPM and whether or not it could be worth learning it.
Recall that Ho et al. [5] saw no difference in sample quality when using a learned variance $\tilde{\beta}_t$ and therefore opted for using the fixed variance β_t.
Nichol et al. [6] came to the same conclusion, finding that as the number of steps T increased (the authors also mentioned that they used $T = 4000$ for their models rather than $T = 1000$ from the original paper), the difference between β_t and $\tilde{\beta}_t$ remained very small. However, they were still interested in the possibility of learning the variance in order to improve the log-likelihood [6].
For this, they used an interpolation approach, where they parametrised the variance as an interpolation between the fixed variance β_t and a learned variance $\tilde{\beta}_t$ in the log domain, with v an output vector from the model containing one component per dimension:

$$\Sigma_\theta(x_t, t) = \exp\left(v \log \beta_t + (1 - v) \log \tilde{\beta}_t\right) \tag{4.1}$$

As the simplified training objective L_{simple} used in the original DDPM paper [5] did not include the variance, Nichol et al. [6] introduced a new training objective L_{hybrid}, defined as:

$$L_{\text{hybrid}} = L_{\text{simple}} + \lambda L_{\text{VLB}} \tag{4.2}$$

with $\lambda = 0.001$ in order to keep L_{simple} as the main training objective [6].

At first, they found that the hybrid objective L_{hybrid} achieved a better log-likelihood and was easier to optimise than just L_{VLB}, contrary to what they had expected [6].
However, they obtained the best log-likelihoods by optimising L_{VLB} directly but with importance samping, which is a technique where samples of a difficult distribution are taken from a distribution that is easier to compute and then the original distribution is approximated by weighted samples [6].

They defined it as:

$$L_{\text{VLB}} = \mathbb{E}_{t \sim p_t}\left[\frac{L_t}{p_t}\right] \qquad \text{where } p_t \propto \sqrt{\mathbb{E}\left[L_t^2\right]} \text{ and } \sum p_t = 1 \tag{4.3}$$

Another change that Nichol et al. [6] made was to use a cosine noising schedule rather than the linear noising schedule used in the original DDPM paper [5].
They explain this change by the fact that towards the end of the forward process, the image is already very noisy and yet more noise is added while not really improving the model, particularly for 32×32 and 64×64 images [6].
In effect, the input images are destroyed by noise very quickly close to the start of the noising process. Nichol and Dhariwal [6] found that up to 20% of the reverse process could be skipped while losing very little image quality and that therefore the last noising steps are redundant.

The cosine schedule they use adds noise at a slower rate throughout the process, in particular at the beginning and the end.

They define the cosine schedule as:

$$f(t) = \cos\left(\frac{\frac{t}{T} + s}{1 + s} \cdot \frac{\pi}{2}\right)^2 \tag{4.4}$$

$$\bar{\alpha}_t = \frac{f(t)}{f(0)} \tag{4.5}$$

$$\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}} \tag{4.6}$$

T is the total number of timesteps, t is the timestep and s is a small offset used to prevent β_t from being too small when $t = 0$ in order to allow the noise ϵ to be predicted more accurately at the beginning of the process [6].

With these changes, Nichol et al. [6] were able to achieve a negative log-likelihood (NLL) of 2.94 bits/dim on CIFAR-10 [24] and 3.53 bits/dim on ImageNet [25], compared to the original DDPM paper [5] which had a NLL of 3.70 on CIFAR-10 and 3.77 on ImageNet [6].

Model	ImageNet	CIFAR
Glow (Kingma & Dhariwal, 2018)	3.81	3.35
Flow++ (Ho et al., 2019)	3.69	3.08
PixelCNN (van den Oord et al., 2016c)	3.57	3.14
SPN (Menick & Kalchbrenner, 2018)	3.52	-
NVAE (Vahdat & Kautz, 2020)	-	2.91
Very Deep VAE (Child, 2020)	3.52	2.87
PixelSNAIL (Chen et al., 2018)	3.52	2.85
Image Transformer (Parmar et al., 2018)	3.48	2.90
Sparse Transformer (Child et al., 2019)	3.44	**2.80**
Routing Transformer (Roy et al., 2020)	**3.43**	-
DDPM (Ho et al., 2020)	3.77	3.70
DDPM (cont flow) (Song et al., 2020b)	-	2.99
Improved DDPM (ours)	**3.53**	**2.94**

Figure 4.1: Comparison of NLLs of different models. Source: [6]

Moreover, Nichol et al. [6] were able to improve sampling speed due to the aforementioned changes, even though they trained their model with $T = 4000$ timesteps.
They were able to train their model with 4000 timesteps, but use only 100 timesteps for sampling which brought them near-optimal FID (Fréchet Inception Distance) scores when using the L_{hybrid} training objective [6].
They had tried to reduce the number of sampling steps with the original fixed variance versions from the DDPM paper [5] but found that the sample quality degraded a lot more [6].

4.2 Denoising Diffusion Implicit Models (DDIMs)

As mentioned previously, the DDPM paper [5] has a forward process and a reverse process which are both Markov processes.
While the forward process can be easily calculated for a specific timestep t with equation (2.10), the reverse process still requires 1000 timesteps in the case of Ho et al.'s [5] model, and Nichol et al.'s model requires 100 and 4000 timesteps for sampling and training respectively [6].
This is time-consuming and computationally expensive compared to GANs which only require one forward pass through the network rather than thousands to produce a sample [9].

To address this issue, Song et al. [9] introduced the Denoising Diffusion Implicit Models (DDIMs) in 2020.
The main objective of DDIMs was to accelerate sampling by making the forward process non-Markovian, which would allow the reverse process to require less iterations [9].

Song et al. [9] define a family $\mathcal{Q}$ of forward processes. Each forward process in $\mathcal{Q}$ is indexed by $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_T\} \in \mathbb{R}^T_{\geqslant 0}$, where σ_t is the standard deviation of the noise added at timestep t:

$$q_\sigma(x_{1:T}|x_0) = q_\sigma(x_T|x_0)\prod_{t=2}^{T} q_\sigma(x_{t-1}|x_t, x_0) \tag{4.7}$$

where

$$q_\sigma(x_T|x_0) = \mathcal{N}(\sqrt{\alpha_T}x_0, (1-\alpha_T)I) \tag{4.8}$$

and for all $t > 1$:

$$q_\sigma(x_{t-1}|x_t, x_0) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot\frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}},\ \sigma_t^2 I\right) \tag{4.9}$$

Song et al. [9] note that when $\sigma_t^2 = 0$, then x_{t-1} is fixed and known in advance.
Therefore, we can see that as Song et al. [9] state, the value of σ controls the stochasticity (or more simply, the randomness) of the forward process.

The authors use Bayes' rule similarly to the original DDPM paper [5] in equation (2.44), but instead to find the forward process $q_\sigma(x_t|x_{t-1}, x_0)$:

$$q_\sigma(x_t|x_{t-1}, x_0) = \frac{q_\sigma(x_{t-1}|x_t, x_0)\ q_\sigma(x_t|x_0)}{q_\sigma(x_{t-1}|x_0)} \tag{4.10}$$

Since $q_\sigma(x_t|x_{t-1}, x_0)$ means that x_t not only depends on x_{t-1} but also on x_0, we see that the forward process is no longer Markovian, which was what the authors were aiming to achieve [9].

However, since the reverse process in the original DDPM paper [5] was an estimation of the forward process which was Markovian, a new reverse process was defined.
Their idea of the reverse process is to sample an image x_T like in the original DDPM paper [5], predict x_0 from x_T and then use this prediction of x_0 to sample x_{T-1} [9].
This is repeated until we reach x_1 and then x_0 is predicted from x_1 [9].
Similar to equation (2.72) from the DDPM paper, Song et al. [9] define the prediction of x_0 from x_t as:

$$f_\theta^{(t)}(x_t) = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\ \epsilon(x_t, t)}{\sqrt{\bar{\alpha}_t}} \tag{4.11}$$

Given this, the reverse process is defined as [9]:

$$p_\theta(x_T) = \mathcal{N}_D(0, I) \tag{4.12}$$

$$p_\theta^{(t)}(x_{t-1}|x_t) = \begin{cases} \mathcal{N}(f_\theta^{(1)}(x_1), \sigma_1^2 I) & \text{if } t = 1 \\ q_\sigma(x_{t-1}|x_t, f_\theta^{(t)}(x_t)) & \text{if } t > 1 \end{cases} \tag{4.13}$$

Therefore, x_{t-1} is sampled from x_t [9]:

$$x_{t-1} = \sqrt{\alpha_{t-1}}\left(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\ \epsilon(x_t, t)}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot\epsilon_\theta(x_t, t) + \sigma_t\epsilon_t \tag{4.14}$$

In order to train the model, Song et al. [9] define the following *variational inference objective*:

$$J_\sigma(\epsilon_\theta) = \mathbb{E}_{q_\sigma(x_{0:T})}[\log q_\sigma(x_{1:T}|x_0) - \log p_\theta(x_{0:T})] \tag{4.15}$$

$$= \mathbb{E}_{q_\sigma(x_{0:T})}\left[\log q_\sigma(x_T|x_0) + \sum_{t=2}^{T} \log q_\sigma(x_{t-1}|x_t, x_0) - \sum_{t=1}^{T} \log p_\theta^{(t)}(x_{t-1}|x_t) - \log p_\theta(x_T)\right] \tag{4.16}$$

This would mean that for different choices of σ, we would have a different objective to optimise [9].
However, the authors [9] prove that J_σ is in fact equivalent to L_γ for certain choices of γ, where L_γ is defined as:

$$L_\gamma(\epsilon_\theta) = \sum_{t=1}^{T} \gamma_t \mathbb{E}_{x_0 \sim q(x_0), \epsilon_t \sim \mathcal{N}(0,I)}\left[\|\epsilon_\theta^{(t)}(\sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon_t) - \epsilon_t\|_2^2\right] \tag{4.17}$$

where γ is a vector of length T with positive values that depend on α.
We easily see that when $\gamma = 1$, we get L_1 which is the training objective from the original DDPM paper [5, 9].
Song et al. prove that

$$\forall \sigma > 0,\ \exists \gamma \in \mathbb{R}_{\geqslant 0}^T \text{ and } \exists C \in \mathbb{R} \text{ s.t. } J_\sigma = L_\gamma + C$$

This means that the optimal solution for J_σ is the same as the optimal solution for L_1, which means that the training objective for DDIMs can be kept the same as the one for DDPMs if the parameters being trained θ are not the same for all timesteps [9].

In order to speed up the sampling process, Song et al. [9] skip some timesteps in the reverse process.
To do this, the forward process is redefined on a subset of timesteps of size S, $\{x_{\tau_1}, \ldots, x_{\tau_S}\}$ and such that $q(x_{\tau_i}|x_0) = \mathcal{N}(\sqrt{\alpha_{\tau_i}}x_0, (1-\alpha_{\tau_i})I)$
Since this is a subset of the timesteps, the training can be done for all timesteps and the sampling can be done on just the subset [9].

With the same amount of training timesteps $T = 1000$ and the same training objective as the DDPM paper [5], Song et al. [9] were able to achieve solid FID scores on CIFAR-10 [24] and CelebA [26] datasets with far fewer sampling timesteps, ranging from 10 to 100 timesteps [9].
A new hyperparameter $\eta \in \mathbb{R}_+$ is introduced to control the interpolation between the DDPM σ and the deterministic σ of the DDIM [9]:

$$\sigma_{\tau_i}(\eta) = \eta\sqrt{\frac{1-\alpha_{\tau_{i-1}}}{1-\alpha_{\tau_i}}}\sqrt{1-\frac{\alpha_{\tau_i}}{\alpha_{\tau_{i-1}}}} \tag{4.18}$$

The results obtained by Song et al. [9] show that the DDIM outperformed the DDPM for 10, 20, 50, 100 and 1000 timesteps and was only slightly worse at 1000 timesteps and $\sigma_{\tau_i} = \hat{\sigma}_{\tau_i} = \sqrt{1-\alpha_{\tau_i}/\alpha_{\tau_{i-1}}}$

		CIFAR10 (32×32)					CelebA (64×64)				
	S	10	20	50	100	1000	10	20	50	100	1000
η	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98
$\hat{\sigma}$		367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**

Figure 4.2: Comparison of FID scores (lower is better) of DDPMs and DDIMs trained on S timesteps, $S < T$ and $\eta = 0$ is DDIM, $\eta = 1$ and $\hat{\sigma}$ are DDPMs. Source: [9]

4.3 Guidance

Recall that when sampling from a DDPM or a DDIM, we start with an image of Gaussian noise x_T and we denoise the image until we get some image x_0 that resembles an image from the original dataset. We may however be interested in sampling something specific; for instance, from some text.
In this case it is helpful to guide the denoising process towards whatever we specified.

4.3.1 Classifier guidance

Classifier guidance was introduced by Dhariwal and Nichol in their *Diffusion Models Beat GANs on Image Synthesis* paper [7] in order to improve the samples generated.
To do this, they trained a classifier $p_\Phi(y \mid x_t, t)$ on noisy images [7] since pre-trained classifiers are generally trained on dataset images and not on noised images.
The reverse process is then guided towards a class y using the gradient $\nabla_{x_t} \log p_\Phi(y \mid x_t, t)$ [7]. The authors prove [7] that if we have a reverse process such as that from the DDPM $p_\theta(x_t, x_{t+1})$, to condition it on class y, we can sample from the following process instead:

$$p_{\theta,\Phi}(x_t \mid x_{t+1}, y) = Z p_\theta(x_t \mid x_{t+1})\, p_\Phi(y \mid x_t) \tag{4.19}$$

where Z is a normalization constant.

Using the score-based formulation, we can easily find the new objective to train the model with classifier guidance [19].
Remember that the score-based generative model learned to estimate $\nabla_{x_t} \log q(x_t)$ (3.21).
If we want to condition the model on class y, we are interested in estimating $\nabla_{x_t} \log q(x_t \mid y)$ [19].
With Bayes' rule, we can rewrite this as [19]:

$$q(x_t \mid y) = \frac{q(y \mid x_t)\, q(x_t)}{q(y)} \tag{4.20}$$

$$\Leftrightarrow \log q(x_t \mid y) = \log \left[\frac{q(y \mid x_t)\, q(x_t)}{q(y)} \right] \tag{4.21}$$

$$\Leftrightarrow \nabla_{x_t} \log q(x_t \mid y) = \nabla_{x_t} \log q(x_t) + \nabla_{x_t} \log q(y \mid x_t) - \nabla_{x_t} \log q(y) \tag{4.22}$$

$$= \nabla_{x_t} \log q(x_t) + \nabla_{x_t} \log q(y \mid x_t) \tag{4.23}$$

We can see that $\nabla_{x_t} \log q(x_t)$ is the original unconditional score from the score-based generative model and $\nabla_{x_t} \log q(y \mid x_t)$ is the gradient of the log-likelihood of the class y given the image x_t [7, 19].
To control the strength of the guidance, the authors introduce a hyperparameter γ that weights the classifier gradient [7, 19]:

$$\nabla_{x_t} \log q(x_t \mid y) = \nabla_{x_t} \log q(x_t) + \gamma \nabla_{x_t} \log q(y \mid x_t) \tag{4.24}$$

Trivially, when $\gamma = 0$, the model doesn't use the classifier and we recover the original score-based generative model.

Dhariwal and Nichol [7] gave two algorithms to train the model with classifier guidance: one for DDPMs and one that also works for DDIMs that uses the score-based generative model formulation:

Algorithm 4 Training a DDPM/DDIM with classifier guidance

Require: *Model* $\epsilon_\theta(x_t, t)$, *Classifier* $p_\Phi(y \mid x_t)$ with y the class label
1: Sample $x_T \sim \mathcal{N}(0, I)$
2: **for** $t = T, T-1, ..., 1$ **do**
3: $\quad \hat{\epsilon} \leftarrow \epsilon_\theta(x_t, t) - \sqrt{1-\bar{\alpha}_t}\, \nabla_{x_t} p_\Phi(y \mid x_t)$
4: $\quad x_{t-1} \leftarrow \sqrt{\bar{\alpha}_{t-1}} \left(\frac{x_t - \hat{\epsilon}\sqrt{1-\bar{\alpha}_t}}{\sqrt{\bar{\alpha}_t}} \right) + \hat{\epsilon}\sqrt{1-\bar{\alpha}_{t-1}}$
5: **end for**
6: **return** x_0

4.3.2 Classifier-free guidance

Despite the impressive performance of the *ablated diffusion model* (ADM) that used classifier guidance, Ho and Salimans made an improvement by using classifier-free guidance [27].
Classifier guidance requires a classifier to be trained on noisy images. This means that not only does it require training a classifier in addition to the generative model, but it also requires the classifier to be trained on noisy images for which pre-trained classifiers do not generally exist [27].
Therefore, Ho and Salimans [27] introduced a classifier-free guidance method which does not require a trained classifier at all.

The authors use an unconditional diffusion model $q(x_t)$ and a conditional diffusion model $q(x_t \mid y)$ where y is the class label [27, 19].

Starting from equation (4.24):

$$\begin{aligned}
\nabla_{x_t} \log q(x_t \mid y) &= \nabla_{x_t} \log q(x_t) + \gamma \nabla_{x_t} \log q(y \mid x_t) && (4.25)\\
&= \nabla_{x_t} \log q(x_t) + \gamma \left(\nabla_{x_t} \log q(x_t \mid y) - \nabla_{x_t} \log q(x_t)\right) && (4.26)\\
&= \nabla_{x_t} \log q(x_t) + \gamma \nabla_{x_t} \log q(x_t \mid y) - \gamma \nabla_{x_t} \log q(x_t) && (4.27)\\
&= \gamma \nabla_{x_t} \log q(x_t \mid y) + \nabla_{x_t} \log q(x_t) - \gamma \nabla_{x_t} \log q(x_t) && (4.28)\\
&= \gamma \nabla_{x_t} \log q(x_t \mid y) + (1-\gamma) \nabla_{x_t} \log q(x_t) && (4.29)
\end{aligned}$$

$\gamma \nabla_{x_t} \log q(x_t \mid y)$ is the score of the conditional model and $(1-\gamma)\nabla_{x_t} \log q(x_t)$ is the score of the unconditional model [19].
However it isn't necessary to train a conditional model and an unconditional model as we can just train the conditional model $q(x_t \mid y)$ and simply ignore the conditioning information y (by setting it to null, for example) to get the unconditional model $q(x_t)$ [27, 19].

5 Further development

5.1 Cascaded Diffusion Models

Training a diffusion model on high-resolution images is computationally expensive and therefore time-consuming.
To address this, Ho et al. [28] introduced the cascaded diffusion model (CDM) which is essentially a pipeline of diffusion models where the first one generates a low-resolution image and the subsequent ones are trained to upsample the image to a higher resolution.
This method is much faster than training a single diffusion model on high-resolution images as most of the important features for sampling are already present in the low-resolution images which are fast to train, whereas training at larger resolutions doesn't add much more detail considering the computation costs [28].

As mentioned, the first model is trained to generate an image as we have seen before. The subsequent models are trained using what Ho et al. call conditioning augmentation, a type of data augmentation where the model is given a lower-resolution image and is trained on modified versions of the image to generate a higher-resolution and overall higher-quality image [28].
For low-resolution images, the authors use added Gaussian noise and use Gaussian blurring for higher resolution images [28].

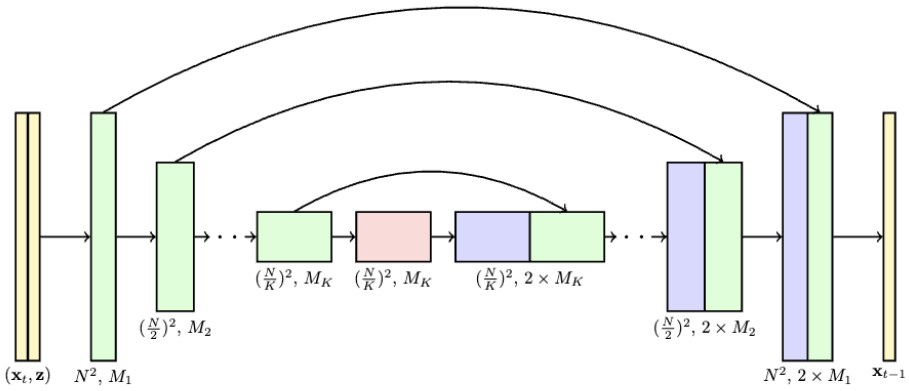

Figure 5.1: U-Net used for each of the model of a CDM pipeline: the upsampling models are given an image to upsample x_t and an upsampled low-resolution image z. For each of the blocks $1, \ldots, K$, the image is upsampled/downsampled by 2, the channels are given by the channel multipliers $M_{1,\ldots,K}$. Source: [28]

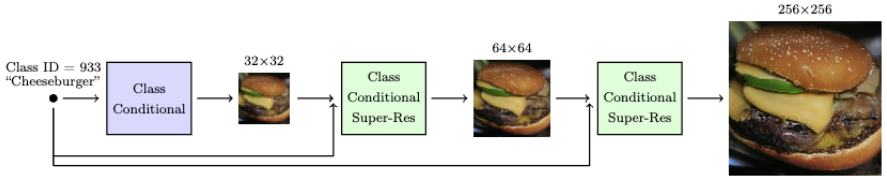

Figure 5.2: Example of a cascading diffusion pipeline for generating a 256×256 resolution image conditioned on a class. Source: [28]

As a result, the authors achieve an FID of 4.88 on ImageNet 256×256 with a CDM, where the first model generates a 32×32 image, the second model upsamples it to 64×64 and the third model upsamples it to 256×256 [28].
As an example, *Imagen* [29] from Google is a popular cascaded diffusion model that can efficiently generate high-resolution images.

5.2 Latent Diffusion Models

Latent Diffusion Models (LDMs) were introduced by Rombach et al. in their paper *High-Resolution Image Synthesis with Latent Diffusion Models* [2].
Rather than do the forward and reverse processes in pixel space, the authors propose to do them in latent space in order to speed up the computation by reducing the dimensionality of the data [2].
In order to do this, a Variational Autoencoder (VAE) [8] is used to first encode the image x into a latent representation z. The forward process is then applied to the latent representation z until we obtain z_T [2]. As for the reverse process, the model (a U-Net in their paper) will denoise the latent representation z_T until we obtain $\tilde{z}$ which is then passed through the decoder of the VAE to obtain the denoised image $\tilde{x}$ [2]. This means that given an image x, the encoder $\mathcal{E}$ will encode x into a latent representation z:

$$z = \mathcal{E}(x) \tag{5.1}$$

Once the forward process and reverse process are done in the latent space, the decoder $\mathcal{D}$ will decode the denoised latent representation $\tilde{z}$ into the denoised image $\tilde{x}$:

$$\tilde{x} = \mathcal{D}(\tilde{z}) \tag{5.2}$$

From the training objective (2.99), we get a new training objective for LDMs [2]:

$$L_{LDM} = \mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t)\|_2^2\right] \tag{5.3}$$

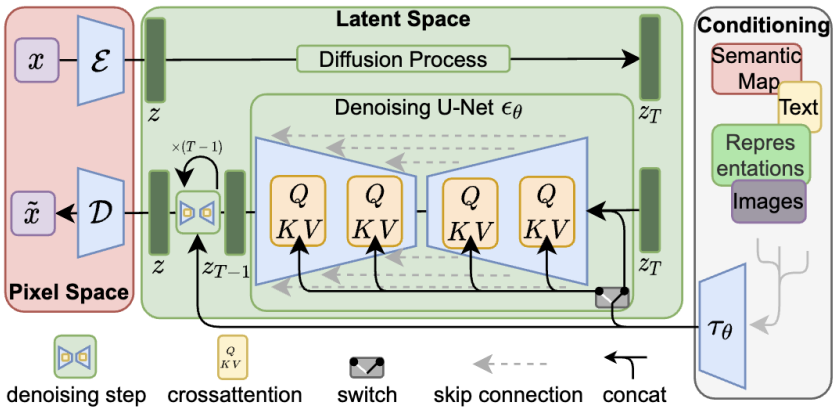

Figure 5.3: Latent Diffusion processes. Source: [2]

In terms of results, Rombach et al. [2] managed to achieve an FID of 3.60 on ImageNet 256 $\times$ 256 with 271 days of training on a Nvidia V100 GPU, which is better than the 3.85 FID achieved by Dhariwal and Nichol [7] with their ADM-G and ADM-U (guidance and upsampling) models which required 349 days of training on a V100 [2].

An example of a current LDM is *Stable Diffusion* [1, 2], which has become very popular nowadays and is easily available for use to the general public.

5.3 Diffusion Transformer

Recently, a new alternative to the U-Net architecture has been proposed by Peebles and Xie in their paper *Scalable Diffusion Models with Transformers* [30].
As the name suggests, the Diffusion Transformer (DiT) architecture is based on the Transformer architecture [15] used for Natural Language Processing (NLP).

The architecture builds upon Latent Diffusion Models (LDMs) [2], since it takes as input the latent representation z.
The first step is called *patchify*: given that we have a spatial input, this input is to be converted into tokens.

Positional encodings are subsequently added to the tokens to give the model information about the spatial structure of the image [30].
This information is then passed through DiT blocks (see figure 5.4) before going through layer normalisation. Finally, the output is passed through a linear layer and then reshaped back in order to return the noise and the covariance matrix Σ [30].

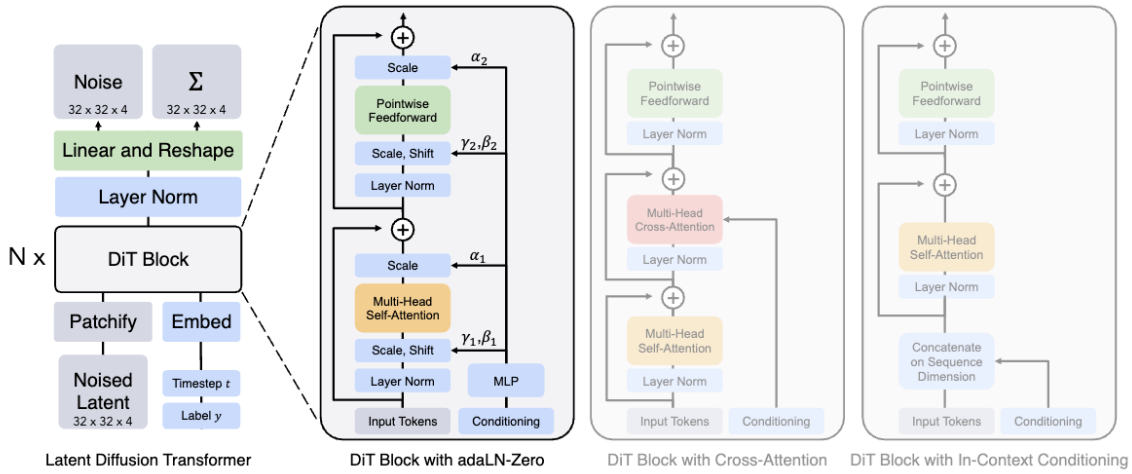

Figure 5.4: Diffusion Transformer architecture. Source: [30]

Compared to the previous experiments on class-conditioned ImageNet 256×256, the DiT model has the best FID of them all with 2.27 [30], showing that the DiT brings significant improvement to sample quality.

Newer models such as *Sora* [3] and *Stable Diffusion 3* use this new DiT architecture and have achieved impressive visual results.

6 Implementation

A simple implementation of a DDPM for the MNIST dataset in PyTorch is given in the Jupyter notebook `DMID.ipynb`. The code aims to be as close as possible to the original DDPM paper [5]. It can be run on a CPU or a CUDA-enabled GPU. The code is available at https://github.com/gregorysedykh/diffusion-models-in-depth.

The code provides an explanation and an implementation of a U-Net architecture similar to the original one. Ho et al. used a self-attention block not only between the two ResNet-like blocks in the bottleneck, but also at the encoder and decoder stages at 16×16 resolution image steps. They trained the model as such for CIFAR-10, CelebA-HQ and LSUN datasets [5].
`DMID.ipynb` contains a model without self-attention blocks as this visually gave worse results on the MNIST dataset, possibly because of overfitting.

The forward process uses the linear noising schedule β and the reverse process uses the same 1000 timesteps from the original DDPM paper [5], however the code is easily modifiable to use a different noising schedule (e.g. cosine noising schedule) and a different number of timesteps. Similarly, a learning rate of 10^{-4} was used as it was found to work well for the MNIST dataset but this can also be changed to the original 2×10^{-4} learning rate.

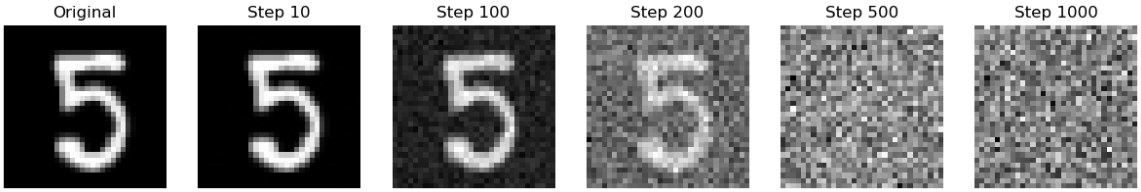

Figure 6.1: Noised images at different timesteps of the forward process

The model contains 7'339'297 parameters, much less than Ho et al.'s CIFAR-10 model which has 35.7 mln parameters and the CelebA-HQ model with its 114 mln parameters [5].
On an Apple M2 Pro chip using MPS, training takes around 70s per epoch for 1000 timesteps and 32×32 MNIST images and sampling 1 image takes around 13.4s.
This means that the model is not so complex but still takes a long time to train and sample from without significant resources such as a TPU v3-8 used by Ho et al. [5].

Figure 6.2: Samples generated by the model

7 Conclusion

Throughout this report, we analysed Denoising Diffusion Probabilistic Models (DDPMs) in detail by covering their mathematical foundation, which includes the two Markovian processes and the simplified training objective. Additionally, we explored the U-Net architecture employed in training these models.
The score-based generative model was also analysed, particularly its use of score matching to estimate the score of the data distribution, and demonstrated its equivalence to DDPMs.
Several improvements to DDPMs were discussed, notably Denoising Diffusion Implicit Models (DDIMs) and the guidance methods, both playing a crucial role in simultaneously improving sample quality and speeding up the sampling process.
Subsequently, we explored the present and the future of diffusion models, especially the Diffusion Transformer (DiT) architecture which is now becoming increasingly popular for models available to the public.
Finally, a straightforward implementation of a DDPM for the MNIST dataset implemented using Python and PyTorch was presented, together with some samples generated by the model.

A Convolution of two Gaussian distributions

Equation (2.7) had given us:

$$x_t = \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \mathcal{N}\left(0, \alpha_t \left(1 - \alpha_{t-1}\right) \mathrm{I}\right) + \mathcal{N}\left(0, \left(1 - \alpha_t\right) \mathrm{I}\right)$$

We can find the distribution of the sum of two independant 1D random variables $X \sim \mathcal{N}\left(\mu_X, \sigma_X^2\right)$ and $Y \sim \mathcal{N}\left(\mu_Y, \sigma_Y^2\right)$ as the convolution of two Gaussian distributions.

Let $Z = X + Y$ with X and Y independent.

$$\begin{aligned}
f_Z(z) &= \int_{-\infty}^{\infty} f_X(x) f_Y(z-x)\, dx \\
&= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left[-\frac{(x-\mu_X)^2}{2\sigma_X^2}\right] \frac{1}{\sqrt{2\pi\sigma_Y^2}} \exp\left[-\frac{(z-x-\mu_Y)^2}{2\sigma_Y^2}\right] dx \\
&= \frac{1}{2\pi\sigma_X\sigma_Y} \int_{-\infty}^{\infty} \exp\left[-\frac{(x-\mu_X)^2}{2\sigma_X^2}\right] \exp\left[-\frac{(z-x-\mu_Y)^2}{2\sigma_Y^2}\right] dx \\
&= \frac{1}{2\pi\sigma_X\sigma_Y} \int_{-\infty}^{\infty} \exp\left[-\frac{(x-\mu_X)^2}{2\sigma_X^2} - \frac{(z-x-\mu_Y)^2}{2\sigma_Y^2}\right] dx \\
&= \frac{1}{2\pi\sigma_X\sigma_Y} \int_{-\infty}^{\infty} \exp\left[-\frac{x^2 - 2\mu_X x + \mu_X^2}{2\sigma_X^2} - \frac{z^2 + x^2 - 2xz - 2z\mu_Y + 2\mu_Y x + \mu_Y^2}{2\sigma_Y^2}\right] dx \\
&= \frac{1}{2\pi\sigma_X\sigma_Y} \int_{-\infty}^{\infty} \exp\left[-\frac{\sigma_Y^2(x^2 - 2\mu_X x + \mu_X^2)}{2\sigma_X^2\sigma_Y^2} - \frac{\sigma_X^2(z^2 + x^2 - 2xz - 2z\mu_Y + 2\mu_Y x + \mu_Y^2)}{2\sigma_X^2\sigma_Y^2}\right] dx \\
&= \frac{1}{2\pi\sigma_X\sigma_Y} \int_{-\infty}^{\infty} \exp\left[-\frac{\sigma_Y^2 x^2 - 2\mu_X\sigma_Y^2 x + \mu_X^2\sigma_Y^2 + \sigma_X^2 z^2 + \sigma_X^2 x^2 - 2xz\sigma_X^2 + 2z\mu_Y\sigma_X^2 - 2\mu_Y x\sigma_X^2 + \sigma_X^2\mu_Y^2}{2\sigma_X^2\sigma_Y^2}\right] dx \\
&= \frac{1}{2\pi\sigma_X\sigma_Y} \int_{-\infty}^{\infty} \exp\left[-\frac{\left(\sigma_X^2 + \sigma_Y^2\right) x^2 - 2x\left(\mu_X\sigma_Y^2 + z\sigma_X^2 - \mu_Y\sigma_X^2\right) + \sigma_X^2 (z-\mu_Y)^2 + \mu_X^2\sigma_Y^2}{2\sigma_X^2\sigma_Y^2}\right] dx
\end{aligned}$$

Let $\sigma_Z = \sqrt{\sigma_X^2 + \sigma_Y^2}$.

$$\begin{aligned}
&= \frac{1}{\sqrt{2\pi}\frac{\sigma_X\sigma_Y}{\sigma_Z}} \frac{1}{\sqrt{2\pi}\sigma_Z} \int_{-\infty}^{\infty} \exp\left[-\frac{x^2 - 2x\frac{\sigma_X^2(z-\mu_Y) + \sigma_Y^2\mu_X}{\sigma_Z^2} + \frac{\sigma_X^2(z-\mu_Y)^2 + \mu_X^2\sigma_Y^2}{\sigma_Z^2}}{2\left(\frac{\sigma_X\sigma_Y}{\sigma_Z}\right)^2}\right] dx \\
&= \frac{1}{\sqrt{2\pi}\frac{\sigma_X\sigma_Y}{\sigma_Z}} \frac{1}{\sqrt{2\pi}\sigma_Z} \int_{-\infty}^{\infty} \exp\left[-\frac{\left(x - \frac{\sigma_X^2(z-\mu_Y) + \sigma_Y^2\mu_X}{\sigma_Z^2}\right)^2 - \left(\frac{\sigma_X^2(z-\mu_Y) + \sigma_Y^2\mu_X}{\sigma_Z^2}\right)^2 + \frac{\sigma_X^2(z-\mu_Y)^2 + \sigma_Y^2\mu_X^2}{\sigma_Z^2}}{2\left(\frac{\sigma_X\sigma_Y}{\sigma_Z}\right)^2}\right] dx \\
&= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\frac{\sigma_X\sigma_Y}{\sigma_Z}} \exp\left[-\frac{\left(x - \frac{\sigma_X^2(z-\mu_Y) + \sigma_Y^2\mu_X}{\sigma_Z^2}\right)^2}{2\left(\frac{\sigma_X\sigma_Y}{\sigma_Z}\right)^2}\right] \\
&\qquad \frac{1}{\sqrt{2\pi}\sigma_Z} \exp\left[-\frac{\sigma_Z^2\left(\sigma_X^2(z-\mu_Y)^2 + \sigma_Y^2\mu_X^2\right) - \left(\sigma_X^2(z-\mu_Y) + \sigma_Y^2\mu_X\right)^2}{2\sigma_Z^2(\sigma_X\sigma_Y)^2}\right] dx
\end{aligned}$$

$$= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\frac{\sigma_X \sigma_Y}{\sigma_Z}} \exp\left[-\frac{\left(x - \frac{\sigma_X^2(z-\mu_Y)+\sigma_Y^2\mu_X}{\sigma_Z^2}\right)^2}{2\left(\frac{\sigma_X\sigma_Y}{\sigma_Z}\right)^2}\right] \frac{1}{\sqrt{2\pi}\sigma_Z} \exp\left[-\frac{(z-(\mu_X+\mu_Y))^2}{2\sigma_Z^2}\right] dx$$

$$= \frac{1}{\sqrt{2\pi}\sigma_Z} \exp\left[-\frac{(z-(\mu_X+\mu_Y))^2}{2\sigma_Z^2}\right] \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\frac{\sigma_X \sigma_Y}{\sigma_Z}} \exp\left[-\frac{\left(x - \frac{\sigma_X^2(z-\mu_Y)+\sigma_Y^2\mu_X}{\sigma_Z^2}\right)^2}{2\left(\frac{\sigma_X\sigma_Y}{\sigma_Z}\right)^2}\right] dx$$

$$= \frac{1}{\sqrt{2\pi}\sigma_Z} \exp\left[-\frac{(z-(\mu_X+\mu_Y))^2}{2\sigma_Z^2}\right]$$

$$= \mathcal{N}\left(z; \mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2\right)$$

Given this, we can refer back to our case and therefore we find:

$$\begin{aligned} x_t &= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \mathcal{N}\left(0, \alpha_t\left(1-\alpha_{t-1}\right) I\right) + \mathcal{N}\left(0, (1-\alpha_t) I\right) \\ &= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \mathcal{N}\left(0, \alpha_t\left((1-\alpha_{t-1}) + 1 - \alpha_t\right) I\right) \end{aligned}$$

References

[1] Stability AI. URL https://github.com/Stability-AI/stablediffusion.

[2] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *CoRR*, abs/2112.10752, 2021, 2112.10752. URL https://arxiv.org/abs/2112.10752.

[3] Tim Brooks, Bill Peebles, Connor Homes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Wing Yin Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators, 2024. URL https://openai.com/research/video-generation-models-as-world-simulators.

[4] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015, 1503.03585. URL http://arxiv.org/abs/1503.03585.

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020, 2006.11239. URL https://arxiv.org/abs/2006.11239.

[6] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *CoRR*, abs/2102.09672, 2021, 2102.09672. URL https://arxiv.org/abs/2102.09672.

[7] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233, 2021, 2105.05233. URL https://arxiv.org/abs/2105.05233.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022, 1312.6114. URL https://arxiv.org/abs/1312.6114.

[9] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *CoRR*, abs/2010.02502, 2020, 2010.02502. URL https://arxiv.org/abs/2010.02502.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015, 1505.04597. URL http://arxiv.org/abs/1505.04597.

[11] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016, 1605.07146. URL http://arxiv.org/abs/1605.07146.

[12] Zhitong Lai, Haichao Sun, Rui Tian, Nannan Ding, Zhiguo Wu, and Yanjie Wang. Rethinking skip connections in encoder-decoder networks for monocular depth estimation, 2022, 2208.13441. URL https://arxiv.org/abs/2208.13441.

[13] Yuxin Wu and Kaiming He. Group normalization, 2018, 1803.08494. URL http://arxiv.org/abs/1803.08494.

[14] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016, 1602.07868. URL http://arxiv.org/abs/1602.07868.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017, 1706.03762. URL http://arxiv.org/abs/1706.03762.

[16] Aapo Hyvarinen. Estimation of non-normalized statistical models by score matching, 2005. URL https://jmlr.org/papers/volume6/hyvarinen05a/hyvarinen05a.pdf.

[17] Pascal Vincent. A connection between score matching and denoising autoencoders, 2010. URL https://www.iro.umontreal.ca/~vincentp/Publications/smdae_techreport.pdf.

[18] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. *CoRR*, abs/1905.07088, 2019, 1905.07088. URL http://arxiv.org/abs/1905.07088.

[19] Calvin Luo. Understanding diffusion models: A unified perspective, 2022, 2208.11970. URL https://arxiv.org/abs/2208.11970.

[20] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *CoRR*, abs/1907.05600, 2019, 1907.05600. URL http://arxiv.org/abs/1907.05600.

[21] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics, 2011. URL https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf.

[22] Yang Song. Blog post: Generative modeling by estimating gradients of the data distribution, 2021. URL https://yang-song.net/blog/2021/score/.

[23] Lilian Weng. What are diffusion models? *lilianweng.github.io*, Jul 2021. URL https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[24] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2024. URL http://www.cs.toronto.edu/~kriz/cifar.html.

[25] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016, 1606.05328. URL http://arxiv.org/abs/1606.05328.

[26] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 12 2015.

[27] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022, 2207.12598. URL https://arxiv.org/abs/2207.12598.

[28] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *CoRR*, abs/2106.15282, 2021, 2106.15282. URL https://arxiv.org/abs/2106.15282.

[29] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022, 2205.11487. URL https://arxiv.org/abs/2205.11487.

[30] William Peebles and Saining Xie. Scalable diffusion models with transformers, 2023, 2212.09748. URL https://arxiv.org/abs/2212.09748.