Denoising Diffusion Probabilistic Model and Image Inpainting

Ibrahim RIDENE Malek BOUHADIDA

Master MVA, ENS Paris-Saclay
Dec 17 2024

GITHUB link of the project: **click on me**

Contents

1 Introduction

This project aims to provide a comprehensive overview of score-based generative models, starting with the concept of score matching as established in [4], which will be discussed in Section 2. This section highlights how denoising autoencoders relate to score matching and introduces key developments in score-based modeling.
Following this, Section 3 will delve into the development of diffusion models, focusing on Denoising Diffusion Probabilistic Models (DDPMs) introduced by Ho et al. in [1]. We will explore how the diffusion framework extends score-based modeling by incorporating a time-dependent noise schedule and a reverse denoising process to generate high-quality samples.
With these theoretical foundations in place, Section 4 presents experiments that demonstrate the practical application of the models, showcasing their performance on benchmark datasets. The results will be critically analyzed in Section 5, where we will discuss these methods' relative strengths and limitations, including trade-offs in computational efficiency and sample quality. Finally, Section 6 provides a conclusion.

2 Denoising Score-Matching

In generative modeling, given data samples $\mathbf{x} = x_1, \ldots, x_n$ drawn from the true data distribution $p(\mathbf{x})$, we usually try to learn a model that allows us to generate new samples. Implicit generative models, such as GANs, are configured to learn the sampling process itself. Explicit models on the other hand, such as DDPMs, try to approximate $p(\mathbf{x})$ by learning $p_\theta(\mathbf{x})$ in order to directly sample from it. In fact, we can represent such a distribution as follows

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

where θ are the learnable parameters of the model, Z_θ is a normalization constant ensuring $\int p(\mathbf{x})\, d\mathbf{x} = 1$, and $f_\theta(\mathbf{x}) \in \mathbb{R}$ is the unnormalized probability model. Such expression is called energy-based.

To fit the model $f_\theta(\mathbf{x})$, we can use maximum-likelihood estimation (MLE). However, to find the parameters θ that maximize the joint density $p_\theta(\mathbf{x})$ we will need to compute the normalization constant Z_θ, which is usually intractable. In such cases, instead of using MLE, a better approach would be to model the **score function** $s(\mathbf{x}) = \nabla_\mathbf{x} \log p_\theta(\mathbf{x})$ by defining a score-based model $s_\theta(\mathbf{x})$ that approximates it. The score here represents the gradient of the log-likelihood of the true data distribution. Specifically, for any given point x in the data space, this score indicates the direction to move in order to reach a region of higher density.

Now, if we insert the definition of the energy-based model in the score function and apply the logarithm quotient rule we get

$$s_\theta(\mathbf{x}) = \nabla_\mathbf{x} \log p_\theta(\mathbf{x}) = \nabla_\mathbf{x} \log \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta} = \nabla_\mathbf{x} \log e^{-f_\theta(\mathbf{x})} - \underbrace{\nabla_\mathbf{x} \log Z_\theta}_{=0} = -\nabla_\mathbf{x} f_\theta(\mathbf{x})$$

We clearly see that the result is independent of the normalizing constant Z_θ and that the score-based model learns a navigation map that, at each point x in the data space, guides us toward regions of higher density.
To train this model, we use **score-matching** which is a simple regression, where $s_\theta(\mathbf{x})$ should learn to best match the gradient of the true log-likelihood at every point x. We can consider the Euclidean distance between the true score and our score-based model, as an objective function

$$J(\theta) = \mathbb{E}_{p(\mathbf{x})}\left[\|s_\theta(\mathbf{x}) - s(\mathbf{x})\|_2^2\right] = \mathbb{E}_{p(\mathbf{x})}\left[\|s_\theta(\mathbf{x}) - \nabla_\mathbf{x} \log p(\mathbf{x})\|_2^2\right]$$

However, this objective function has two main issues. First, it requires access to the true score of the data distribution, which is typically unavailable. Second, it struggles to learn accurate estimates of the score in low-density regions, as these regions are weighted by $p(\mathbf{x})$ when computing the expectation $\mathbb{E}_{p(\mathbf{x})}$.
To address these problems, we can use the link between score matching and the Denoising Autoencoder (DAE) objective [4].

The purpose of DAEs is to learn a model that denoises a corrupted sample $\tilde{\mathbf{x}}$ to be able to return to the original sample $\mathbf{x}$. We can create a perturbed version of our original dataset by adding Gaussian noise ϵ to each data sample such that the corrupted version of $\mathbf{x}$ is defined as $\tilde{\mathbf{x}} = \mathbf{x}+\epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2 \cdot \mathbf{I})$.

We define the explicit score-matching objective for the corrupted dataset $q_\sigma(\tilde{\mathbf{x}})$ as follows

$$J_{\text{explicit}}(\theta) = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\frac{1}{2}\|s_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})\|_2^2\right].$$

which is equivalent to

$$J_{\text{DSM}_\sigma}(\theta) = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}},\mathbf{x})}\left[\frac{1}{2}\|s_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2\right]$$

where $q_\sigma(\tilde{\mathbf{x}}, \mathbf{x}) = q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})$.
In fact, $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$ is a complex distribution so we cannot evaluate it. However, $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ follows a normal distribution $\mathcal{N}(\mathbf{x}, \sigma^2 \cdot \mathbf{I})$ as

$$\tilde{\mathbf{x}} = \mathbf{x} + \epsilon \iff \tilde{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, \sigma^2 \cdot \mathbf{I})$$

with $\epsilon \sim \mathcal{N}(0, \sigma^2 \cdot \mathbf{I})$.
This allows us to derive the following gradient

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

More details about the calculations can be found in the Appendix.
Intuitively, the gradient corresponds to the direction of moving from $\tilde{\mathbf{x}}$ back to the original $\mathbf{x}$ (i.e. denoising it), and we want our score-based model $s_\theta(\tilde{\mathbf{x}})$ to match that as best as it can.

While this method seems to be very promising, there exists in fact a trade-off between small versus large additive noise scales σ^2. With small noise scales the denoising score approximates the ground truth score well, but we cannot cover much of the low density regions, whereas with larger noise scales we can cover more of the low density regions, but our targets are less accurate, as they may not match the actual score anymore.
To bypass this problem, we can consider different noisy versions of our dataset, using different noise scales. We populate the whole data space with a large noise scale so that in low density regions we get a rough direction in which to move. Then, iteratively, we decrease our noise scale and get a finer and finer direction towards regions with higher density. This is the intuition behind DDPMs.

3 Denoising Diffusion Models

Diffusion models operate on the principle of transforming data through a series of learned Gaussian transitions starting from $p(x_T) = \mathcal{N}(x_T; 0; I)$, forming a Markov chain known as the reverse process. This process is mathematically expressed as:

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

The **forward process** on the other hand, expressed as $q(x_{1:T}|x_0)$, is also a Markov chain structure in which Gaussian noise is incrementally added to the data over time according to a variance schedule $\beta_1, \ldots, \beta_T$:

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

A key feature of this recursive formulation is that it allows the direct sampling of x_t at any timestep t, with a fixed variance β_t, using:

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

where $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ and $\alpha_t = 1 - \beta_t$.
In fact, sampling from $q(x_t|x_0)$ is performed via the reparametrization trick. Let $\mathbf{z} \sim \mathcal{N}(0, \sigma^2 I)$, the data $\mathbf{x}$ is first standardized using $\tilde{\mathbf{z}} = \frac{\mathbf{x}-\mu}{\sigma}$, where $\tilde{\mathbf{z}} \sim \mathcal{N}(0, I)$. And then, to reconstruct $\mathbf{x}$, we perform the inverse operation of the normalization, that involves scaling $\tilde{\mathbf{z}}$ by σ and shifting it by μ to obtain:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\tilde{\mathbf{z}}_t,$$

where $\tilde{\mathbf{z}}_t \sim \mathcal{N}(0, I)$.

Now, to generate new instances from $p(x_0)$, we start from $x_T \sim \mathcal{N}(0, I)$ and reverse the process. A neural network, generally a UNet, is trained to approximate the parameters of the distribution $p(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$, specifically predicting the mean $\mu_\theta(x_t, t)$ and the covariance $\Sigma_\theta(x_t, t)$ from the noisy image x_t at timestep t.
In fact, directly maximizing $p_\theta(x_0)$ for each x_0 is not computationally feasible due to the complexity of aggregating all reverse trajectories. Instead, we minimize the variational lower bound on the negative log-likelihood to effectively train the network.

The training optimizes the variational lower-bound on negative log-likelihood:

$$\mathcal{L}_{\text{vlb}} = -\log p_\theta(x_0|x_1) + \text{KL}\left(p(x_T|x_0) \,\|\, \pi(x_T)\right) + \sum_{t>1} \text{KL}\left(p(x_{t-1}|x_t, x_0) \,\|\, p_\theta(x_{t-1}|x_t)\right)$$

focusing on the closeness of $p_\theta(x_{t-1}|x_t)$ to the true posterior of the forward process.
Ho *et al.* [1] propose a specific parameterization for $\mu_\theta(x_t, t)$, leading to:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}x_t - \sqrt{\frac{\beta_t}{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)$$

The simplified objective for denoising score matching becomes:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t\sim[1,T]}\mathbb{E}_{x_0\sim p(x_0)}\mathbb{E}_{\epsilon_t\sim\mathcal{N}(0,I)}\left[\|\epsilon_t - \epsilon_\theta(x_t, t)\|^2\right]$$

where the network predicts noise.

4 Experiments

In this section, we detail our experimental analysis, exploring the effects of different types of scheduler, various hyperparameter configurations, and two distinct neural network architectures (based on U-Net) used in the reverse process. We also outline the sampling algorithm used to generate new data instances, as well as implementing a RePaint algorithm based on [2] by Andreas Lugmayr, both as an application of the trained DDPM. While in [1], the authors focused on the CIFAR10 and ImageNet datasets, we will be focusing on the MNIST, MNIST-Fashion, CIFAR10 and CelebA datasets.

4.1 Model architecture

In this project, we developed a diffusion model with the following components:

- A forward method to transform the input image into a noisy image by injecting progressively a small amount of noise with respect to the chosen noise schedule.
- A U-Net model to predict the injected noise from the noisy image. We developed a simplified U-Net architecture as well as a more sophisticated version.
- A Sampling method that enables us to generate new samples based on a pure noise input. The generated samples share similar distribution with the training data.
- An InPaint model designed to reconstruct the masked regions of an image using an already pre-trained diffusion model.

This model architecture was trained on several datasets.
In the next sections, we will illustrate the different training conditions of the model, as well as the application of the diffusion model.

4.2 Selecting schedulers

As detailed in Section 3, the forward process in a DDPM involves incrementally adding Gaussian noise to data over a series of timesteps, governed by a variance schedule $\beta_1, \beta_2, \ldots, \beta_T$. This process is characterized by the following equations:

- **Forward Diffusion Process**:

  At each timestep t, the data point $\mathbf{x}_{t-1}$ is transformed into $\mathbf{x}_t$ by adding Gaussian noise

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

- **Direct Sampling at Arbitrary Timestep**:

  Due to the properties of Gaussian distributions, it is possible to sample $\mathbf{x}_t$ at any arbitrary timestep t directly from the original data point $\mathbf{x}_0$ without iterating through all previous timesteps

  $$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

  where $\bar{\alpha}_t$ is defined as:

  $$\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s).$$

This formulation ensures that, as t increases, the data progressively transforms into a Gaussian noise distribution, facilitating the training of the diffusion model.

The scheduling of betas β_t is critical as it directly controls the amount of noise added to the data at each timestep during the forward diffusion process. These parameters determine how the data transitions from its original state to pure Gaussian noise over the course of T timesteps. A well-designed noise schedule ensures a smooth and gradual addition of noise, allowing the model to learn to reverse this process effectively during training. In the reverse process, betas also influence the reconstruction quality by governing the variance of the denoising steps.
In [1], the authors chose the β_t schedule to be linear from $\beta_1 = \beta_{min} = 0.0001$ to $\beta_T = \beta_{max} = 0.02$ where $T = 1000$. In this project, we have chosen to explore and compare different types of schedulers, namely a linear scheduler, a cosine scheduler, a quadratic scheduler, an exponential scheduler, and finally a logarithmic scheduler. Each scheduler contributes differently on the amount of noise added eat each timestep for both the forward and backward processes. The exact formulas of the schedulers can be found in the Appendix.

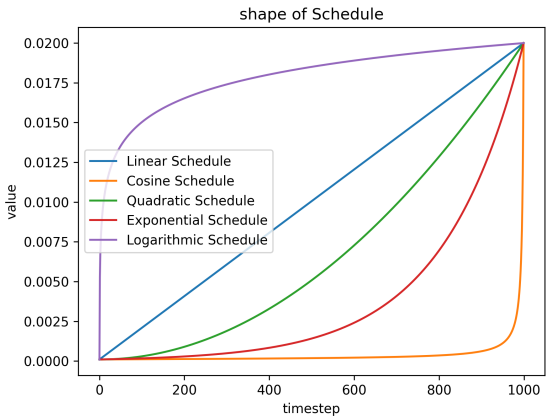
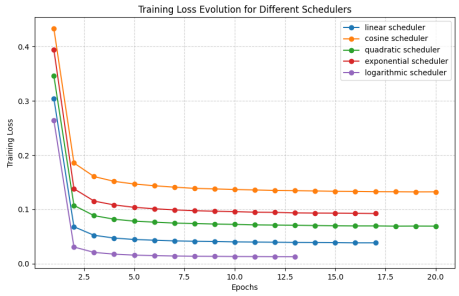


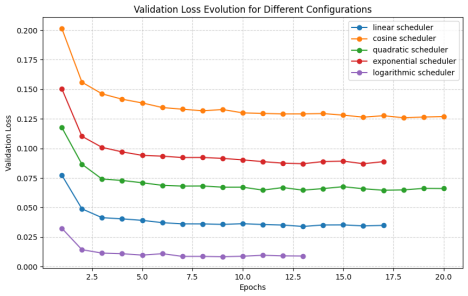
Figure 1: Shapes of the five schedulers as a function of the TimeSteps

In order to compare between the different schedulers, we trained a series of diffusion models under the default configurations proposed in [1]. Specifically, we set the total number of diffusion steps to $T = 1000$, with the variance schedule defined by $\beta_{\min} = 0.0001$ and $\beta_{\max} = 0.02$.

To ensure a fair comparison, all models were trained using the same dataset $CIFAR10$ and training configuration. These choices were motivated by the need to explore how different noise schedules impact the performance of Denoising Diffusion Probabilistic Models (DDPMs) under standardized settings.



(a) Evolution of the training MSE for different schedulers



(b) Evolution of the validation MSE for different schedulers

Figure 2: Evolution of the different schedulers during training and validation

As we can see, the **linear and logarithmic** schedulers achieve good training and validation losses, thus we will focus only on these two schedulers for all the next steps and do a comparison between them. Note that in our implementation, we added an early stopping criteria to avoid overfitting, which is the reason why we are not having complete curves for some schedulers during the 20 epochs.

4.3 Hyperparameter Tuning

Now that we have selected the most promising noise schedulers, we move to the hyperparameter tuning step in order to increase our diffusion model's performance. As explained in previous sections, we have the number of time steps T, the minimum and maximum values of the noise scheduler β_{min} and β_{max} that need to be tuned. There are in fact other hyperparameters namely the batch size and the number of epochs, however, for computational purposes, we chose to keep them fixed.
Since the learning rate is a critical hyperparameter, we implemented a learning rate scheduler that adjusts the learning rate during training to improve convergence. It starts with a low learning rate, gradually increases to a peak (max_lr) during the first phase of training (proportional to pct_start), and then decreases to a very low value by the end of training. Simultaneously, momentum is adjusted inversely, starting high, decreasing as the learning rate rises, and increasing again during the cooldown phase. This strategy helps the model converge faster and potentially avoids overfitting.

To find the best combination of T, β_{min} and β_{max} we used bayesian optimization [3]. This approach uses a probabilistic surrogate model to guide the search process of the hyperparameters. It in fact balances the exploration of unexplored regions with the exploitation of known promising areas, reducing the number of evaluations needed to identify the best set of parameters, which is crucial in our case, as the training of complex neural architectures is computationally expensive.

This bayesian model provides a posterior distribution over possible functions based on prior observations, capturing both the predicted mean $\mu(x)$ and uncertainty $\sigma(x)$ at each point x. The optimization process iteratively selects the next evaluation point by maximizing an acquisition function $\alpha(x)$, which balances exploration of uncertain regions and exploitation of areas with high predicted values. A common acquisition function is the Expected Improvement (EI), defined as:

$$\alpha_{\text{EI}}(x) = \mathbb{E}[\max(0, f(x^+) - f(x))]$$

where $f(x^+)$ is the best observed value so far. By iteratively evaluating the objective function at points that maximize $\alpha(x)$, Bayesian optimization efficiently converges to the global optimum of $f(x)$.

The number of trials (exploitation/exploration steps) is fixed to 5 in our implementation of the Bayesian optimization based on Optuna library. We executed the hyperparameters tuning pipeline for both linear and logarithmic schedulers, and for 3 types of datasets (MNIST, FashionMNIST and CIFAR10). Each combination of scheduler/dataset gives different results of hyperparameters that maximize the objective function based on the validation loss.
This table resumes the final considered hyperparameters:

Dataset	**T**	**beta min**	**beta max**	**Best Loss**
Fashion	1757	0.0007645481	0.0328679383	0.01454419
CIFAR	1132	0.0004989979	0.0387503176	0.02055065
MNIST	1722	0.0000252098	0.0479183811	0.01064030

Table 1: Results of Hyperparameter tuning (Linear scheduler)

Dataset	**T**	**beta_min**	**beta_max**	**Best Loss**
Fashion	1927	1.1146056e-05	0.041009289	0.00310464
CIFAR	1931	6.4127417e-05	0.042472445	0.00290811
MNIST	1931	0.0003399997	0.039108512	0.00201147

Table 2: Results of Hyperparameter tuning (Logarithmic scheduler)

4.4 Sampling process

The sampling process in DDPM (reverse diffusion process) enable us to generate samples based on the trained diffusion model with respect to the training data. It involves progressively denoising a sample of pure Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ back into a coherent data sample $\mathbf{x}_0$. Given that the forward process adds Gaussian noise step-by-step, the reverse process approximates the true data distribution by learning the conditional probabilities $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ at each timestep t. As explained in previous sections, this reverse process is modeled as a Markov chain with Gaussian transitions parameterized by a neural network:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)),$$

where $\mu_\theta(\mathbf{x}_t, t)$ is the predicted mean and $\Sigma_\theta(\mathbf{x}_t, t)$ is the variance, both learned by a neural network during training. Typically, the variance is fixed or simplified, while the mean is derived from the noise prediction network. Starting from pure noise $\mathbf{x}_T$, the reverse process iteratively applies the denoising steps until $\mathbf{x}_0$ is reached

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z},$$

where $\epsilon_\theta(\mathbf{x}_t, t)$ is the predicted noise at timestep t, $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ is the cumulative product of noise coefficients, σ_t is the standard deviation of the noise at step t, and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a Gaussian random variable. This iterative sampling process gradually removes noise, reconstructing the data sample $\mathbf{x}_0$ with high fidelity.

After training our diffusion model, we generated samples per schedule per datasets. The generated samples can be found in the Appendix 3.

4.5 InPainting process

As an application of DDPMs, we implemented a RePaint model in order to generate the masked regions of an input image. This model is based on the paper [2] by Andreas Lugmayr.
In the RePaint framework, the goal is to reconstruct masked regions of an image while preserving the known regions. This is achieved by leveraging a diffusion model to iteratively sample and refine the inpainted image. The process begins with an initial corrupted image $\mathbf{x}_0$, where the known regions are intact, and the masked regions require reconstruction. The RePaint method involves a sampling-like

process in which both forward and reverse steps are performed iteratively. At each timestep t, Gaussian noise is added to the known regions, following the forward diffusion process:

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

where β_t is the variance schedule. This ensures that the known regions remain consistent with the noise profile required by the diffusion model. Simultaneously, the masked regions undergo denoising using the reverse process:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)).$$

Here, the model predicts the distribution of the previous state $\mathbf{x}_{t-1}$ based on the current state $\mathbf{x}_t$. After each timestep, the known regions in the reconstructed image are replaced with their original, uncorrupted values to maintain consistency. This injection of noise into the known regions and simultaneous denoising of the masked regions are repeated until the process converges at timestep $t = 0$. The result is an inpainted image where the masked regions seamlessly blend with the original known areas.

We implemented a RePaint model based on the trained diffusion model and the algorithm from [2] that can be found in the appendix 5
We tested our model on an image per dataset. The implemented mask will mask the upper half of the picture. We got interesting results 4

4.6 UNet with Attention

In all above experiments, where we trained and used the diffusion model for sampling and Repainting, the implemented model to predict noise is based on a simplified U-Net architecture that includes Convolutional layers, residual blocks, DownSampling and UpSampling blocks. In this section, we tried to develop a better architecture for U-Net by injecting many attention blocks between DownSampling/UpSampling blocks and the Residual blocks.

Integrating attention blocks into the U-Net architecture significantly enhances the model's capacity to capture long-range spatial dependencies. While traditional convolutions in U-Net perform well at capturing local patterns due to their limited receptive field, they struggle to encode relationships between distant regions of an image. Attention blocks address this limitation by enabling the model to refine features across the entire spatial domain through self-attention mechanisms. By allowing each spatial location in the feature map to interact with all other locations, attention blocks improve the model's ability to understand global context, which is essential for tasks such as image synthesis, denoising, and inpainting.
The results of the diffusion model based on U-Net with attention is promising as we get lower validation loss after tuning the hyperparameters compared to the model based on U-Net without attention.
We made a comparison between both models for the RePaint task based on 3 different pictures selected from the test set of the CIFAR10 dataset 6.

5 Discussion

5.1 Linear vs Logarithmic Scheduler

Based on the results of Figure 2, we selected previously the log and linear schedulers. throughout all our implementation process, we made a comparison between both schedulers on important tasks such as sample generation and image inpainting.

5.1.1 Sample Generation

Based on the results of Figure 3, we can easily notice that, despite a lower validation MSE loss for the logarithmic scheduler, the results on sampling tasks show that the logarithmic scheduler introduces more noise in the generated samples compared to the linear scheduler due to the way noise is distributed across the timesteps during the forward diffusion process.

In the logarithmic scheduler, a large amount of noise is injected at the very early timesteps, which rapidly corrupts the input data. This initial large noise injection causes significant destruction of fine-grained details in the input, leading to the loss of subtle features that may be critical for generating high-quality samples. As the process progresses, the logarithmic scheduler adds very small amounts of noise in later steps, meaning that the model has limited opportunity to gradually refine and recover the lost details during the reverse denoising process.

On the other hand, the linear scheduler distributes noise more uniformly across all timesteps. This gradual and consistent noise addition allows the model to retain more structure and details of the data, as the reverse denoising process can progressively refine the samples. The even noise distribution avoids overwhelming the model early in the process and provides a smoother transition between timesteps, leading to cleaner and more coherent generated samples.

Thus, the logarithmic scheduler's aggressive noise injection at the start creates challenges during the reverse process, as the model must recover from heavy corruption with fewer opportunities for fine-tuning. This results in noisier and less accurate samples compared to the linear scheduler, which offers a more balanced noise schedule conducive to gradual denoising and reconstruction.

5.1.2 Image Inpainting

In the image inpainting task results in Figure 4, the logarithmic scheduler introduces more noise in the inpainted regions compared to the linear scheduler. Here, we can make the same analysis as the sample generation task. This problem is due to the large noise injected at the initial steps, making it harder to recover fine details and maintain consistency with the known regions during the reverse process. In contrast, the linear scheduler's gradual noise addition allows for smoother and more coherent inpainting results.

5.2 U-Net with and without Attention

By analyzing the results of the image inpainting task in Figure 6, we can clearly notice that the generation of the masked regions of the image respects more the global distribution for known regions in UNet with Attention.

As a result, comparing the U-Net architecture with and without attention blocks highlights the significant impact of attention mechanisms on the quality of the generated images. The UNet without attention relies solely on local convolutions, which effectively capture fine-grained details but struggle to model long-range dependencies across the image. As a result, the inpainted regions may lack global coherence, leading to inconsistencies between the reconstructed masked areas and the unmasked regions.

In contrast, the UNet with attention blocks introduces a self-attention mechanism that captures both local features and long-range dependencies within the image. This allows the model to better understand the global structure and context of the image during the denoising process. The results demonstrate that the attention-enhanced UNet provides superior inpainting performance, particularly in scenarios where preserving both fine details and global structure is critical.

5.3 Improvements

In this section, we discuss two key improvements that can enhance our current implementation of the DDPM-based image inpainting framework.

First, incorporating the **Fréchet Inception Distance** (FID) score during training can provide a better evaluation of the generated images' quality with respect to the global distribution of the data. While the **Mean Squared Error** (MSE) focuses on minimizing pixel-level differences and preserving local details, combining the FID score with the MSE can ensure that the model optimizes for both local consistency and global coherence.

Second, we propose extending the training, sampling, and inpainting processes to incorporate **class labels**. For the sampling task, conditioning the model on a specific class label would allow the generation of samples belonging to a desired category, enabling more control over the output. Similarly,

for the image inpainting task, providing the class label during training and inference would guide the model to generate masked regions that are not only consistent with the known parts of the image but also belong to the same class. This added conditioning can encourage the model to be more **creative** in reconstructing the masked regions while ensuring that the generated content aligns semantically with the rest of the image, thus improving the overall quality and coherence of the inpainting results.

These improvements aim to refine the model's performance both in terms of **image quality** and **control** over the generation process, addressing limitations observed in the current implementation.

6 Conclusion

In this project, we successfully implemented a DDPM from scratch, focusing on both the theoretical and practical aspects of the framework. We introduced a simplified version of the U-Net architecture to reduce computational complexity while maintaining performance. To enhance the model's effectiveness, we explored different noise schedulers and performed hyperparameter tuning to optimize the training process.

Furthermore, we demonstrated the model's capabilities through sample generation and image inpainting tasks, showcasing its ability to produce high-quality and coherent results. To address the limitations of standard U-Net, we developed an improved version by integrating attention mechanisms, enabling the model to better capture long-range dependencies and refine global structures in the images.

These contributions collectively advance the understanding and implementation of DDPMs, providing a solid foundation for future work on diffusion-based generative models and their applications.

7 Contributions

GITHUB link of the project: **click here**

Ibrahim RIDENE:

- Implementation of the diffusion model and simplified U-Net architecture from scratch
- Implementation of the different schedulers, as well as the scheduler tuning pipeline
- Implementation of the sampling method to generate samples, as well as the sampling pipeline
- Implementation of the Inpaint model to perform image inpainting based on trained models, as well as the Inpainting pipeline

Malek BOUHADIDA:

- Implementation of the dataset class to integrate the different datasets
- Implementation of the training and testing pipelines to train and validate the different diffusion models based on dataset type.
- Implementation of the Hyperparameters tuning pipeline based on Bayesian Optimization
- Implementation of the modified U-Net model by integrating Attention blocks

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *arXiv preprint* abs/2006.11239 (2020).

[2] Andreas Lugmayr et al. "RePaint: Inpainting using Denoising Diffusion Probabilistic Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[3] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems*. Vol. 25. 2012, pp. 2951–2959.

[4] Pascal Vincent. "A connection between score matching and denoising autoencoders". In: *Neural Computation* 23 (2011), pp. 1661–1674.

8 Appendix

8.1 Gradient calculation details

$$\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \nabla_{\tilde{\mathbf{x}}} \log \mathcal{N}(\mathbf{x}, \sigma^2 \cdot \mathbf{I}) = \nabla_{\tilde{\mathbf{x}}} \log \frac{\exp\left(-\frac{1}{2}(\tilde{\mathbf{x}}-\mathbf{x})^T(\sigma^2\mathbf{I})^{-1}(\tilde{\mathbf{x}}-\mathbf{x})\right)}{\sqrt{(2\pi)^d|\sigma^2\mathbf{I}|}}$$

$$= \nabla_{\tilde{\mathbf{x}}} \log \exp\left(-\frac{1}{2}(\tilde{\mathbf{x}}-\mathbf{x})^T(\sigma^2\mathbf{I})^{-1}(\tilde{\mathbf{x}}-\mathbf{x})\right) - \underbrace{\nabla_{\tilde{\mathbf{x}}} \log \sqrt{(2\pi)^d|\sigma^2\mathbf{I}|}}_{=0}$$

$$= -\frac{1}{2\sigma^2}\nabla_{\tilde{\mathbf{x}}}(\tilde{\mathbf{x}}-\mathbf{x})^T\mathbf{I}(\tilde{\mathbf{x}}-\mathbf{x}) = -\frac{1}{2\sigma^2}\nabla_{\tilde{\mathbf{x}}}(\tilde{\mathbf{x}}-\mathbf{x})^2 = -\frac{(\tilde{\mathbf{x}}-\mathbf{x})}{\sigma^2} = -\frac{(\mathbf{x}-\tilde{\mathbf{x}})}{\sigma^2}.$$

8.2 Scheduler formulas

We define the noise schedulers as follows, where $\beta_{\min}$ and $\beta_{\max}$ are the minimum and maximum noise levels, respectively

- **Linear Scheduler:** $\beta_t = \beta_{\min} + t \cdot \frac{\beta_{\max}-\beta_{\min}}{T}$
- **Cosine Scheduler:** $\beta_t = \beta_{\min} + \left(\frac{\cos\left(\frac{t}{T}+s\right)}{\cos(s)}\right)^2 \cdot (\beta_{\max}-\beta_{\min})$ where s is a small stability constant
- **Quadratic Scheduler:** $\beta_t = \beta_{\min} + (\beta_{\max}-\beta_{\min}) \cdot \left(\frac{t}{T}\right)^2$
- **Exponential Scheduler:** $\beta_t = \beta_{\min} \cdot \left(\frac{\beta_{\max}}{\beta_{\min}}\right)^{t/T}$
- **Logarithmic Scheduler:** $\beta_t = \beta_{\min} + (\beta_{\max}-\beta_{\min}) \cdot \frac{\log(1+c\cdot t)}{\log(1+c\cdot T)}$ where c is a scaling constant

8.3 Figures

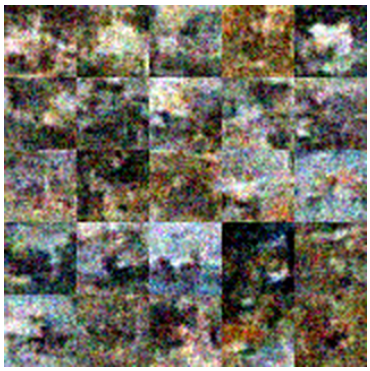
(a) MNIST with Linear schedule

(b) MNIST with Log schedule

(c) FashionMNIST with Linear schedule

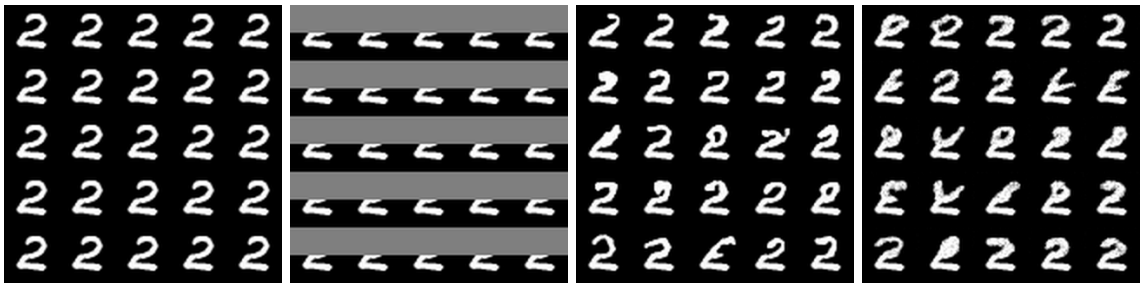
(d) FashionMNIST with Log schedule

(e) CIFAR10 with Linear schedule

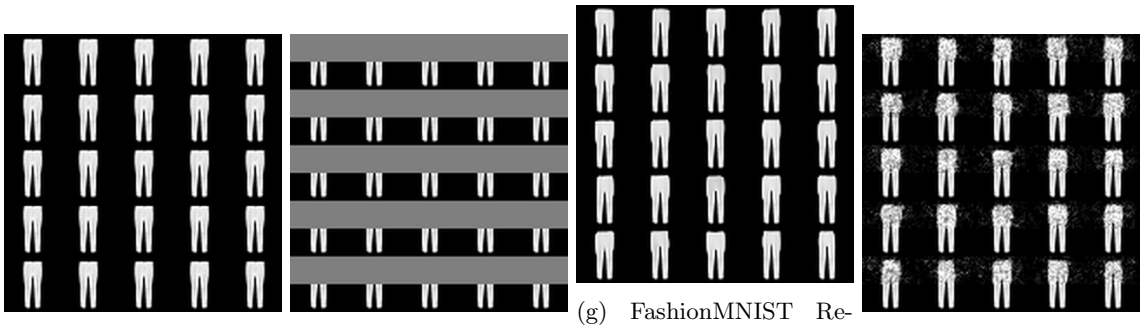


(f) CIFAR10 with Log schedule

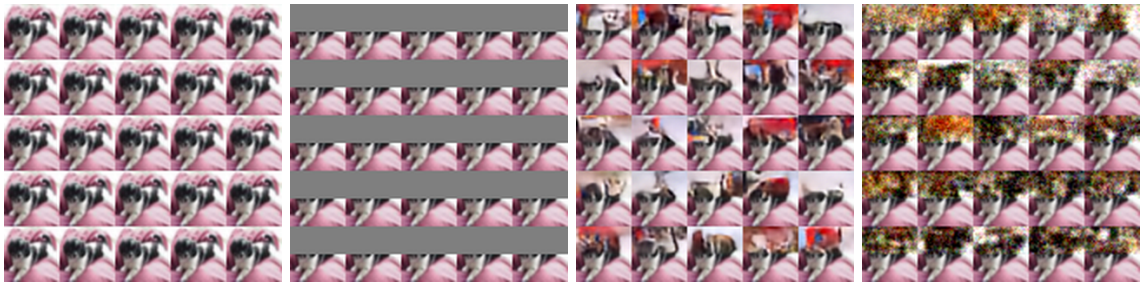
Figure 3: Comparison between samples generated based on Linear and Logarithmic schedulers



(a) MNIST original sample (b) MNIST masked sample (c) MNIST RePainted (Linear Scheduler) (d) MNIST RePainted (Log Scheduler)



(e) FashionMNIST original sample (f) FashionMNIST masked sample (g) FashionMNIST RePainted (Linear Scheduler) (h) FashionMNIST RePainted (Log Scheduler)



(i) CIFAR10 original sample (j) CIFAR10 masked sample (k) CIFAR10 RePainted (Linear Scheduler) (l) CIFAR10 RePainted (Log Scheduler)

Figure 4: Comparison between samples generated based on Linear and Logarithmic schedulers

Algorithm 1 Inpainting using our RePaint approach.

$x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  for $u = 1, \dots, U$ **do**
    $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\epsilon = \mathbf{0}$
    $x_{t-1}^{\text{known}} = \sqrt{\bar{\alpha}_t} x_0 + (1 - \bar{\alpha}_t)\epsilon$
    $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
    $x_{t-1}^{\text{unknown}} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$
    $x_{t-1} = m \odot x_{t-1}^{\text{known}} + (1 - m) \odot x_{t-1}^{\text{unknown}}$
    if $u < U$ and $t > 1$ **then**
      $x_t \sim \mathcal{N}(\sqrt{1 - \beta_{t-1}} x_{t-1}, \beta_{t-1}\mathbf{I})$
    end if
  end for
end for
return x_0

Figure 5: Shapes of the five schedulers as a function of the TimeSteps

(a) CIFAR10 original sample (b) CIFAR10 masked sample (c) CIFAR10 RePainted (UNET) (d) CIFAR10 RePainted (UNET with attention)

(e) CIFAR10 original sample (f) CIFAR10 masked sample (g) CIFAR10 RePainted (UNET) (h) CIFAR10 RePainted (UNET with attention)

(i) CIFAR10 original sample (j) CIFAR10 masked sample (k) CIFAR10 RePainted (UNET) (l) CIFAR10 RePainted (UNET with attention)

Figure 6: Comparison between samples generated based on UNET with and without attention blocks (with linear scheduler)