M2: Application of Machine Learning

Coursework Assignment

University of Cambridge

Department of Physics

Prepared by:
Sabahattin Mert Daloglu

Word Count: 2979

March 28, 2024

1 Training a Diffusion Model

1.0.1 1(a)

Denoising Diffusion Probabilistic Models (DDPM) refine classical diffusion models by integrating denoising score matching with the Langevin dynamics. This approach, which frames the reverse process as a task of denoising, enables the generation of high-quality samples. The underlying principles of diffusion models are foundational to understanding DDPMs. These models are characterized by two primary processes: the forward and reverse diffusions. The forward process slowly introduces Gaussian noise into the data, *diffusion* concept inspired by statistical mechanics, functioning as an encoder. It is a Markov Chain process that slowly introduces noise to transform data distribution into a unit Gaussian, with pre-set noise schedules determined before training. The reverse diffusion, in contrast, is where learning occurs, by using probabilistic mappings from latent samples back to the original data. A key point in DDPM is the minimal noise addition at each step, allowing the reverse denoising to be represented as a Gaussian distribution. Thus, a neural network predicts the noise at any given step, allowing for the reconstruction of high-quality images by processing sampled noise through these learned dynamics[5]. A few important nomenclature along with important equations for the discussion of DDPM algorithms are given in the next part.

For the forward process, we define the sample $\boldsymbol{x}$ being mapped through a series of latent variables $\boldsymbol{z}_1, \boldsymbol{z}_2, ...\boldsymbol{z}_T$, according to:

$$z_1 = \sqrt{1-\beta_1} \cdot x + \sqrt{\beta_1} \cdot \varepsilon_1 \tag{1}$$

$$\mathbf{z}_t = \sqrt{1-\beta_t} \cdot \mathbf{z}_{t-1} + \sqrt{\beta_t} \cdot \varepsilon_t \quad \forall t \in \{2, \dots, T\}, \tag{2}$$

The steps are controlled by noise schedule hyperparameters $\beta_t \in [0, 1]$ and with the noise term, ε_t, drawn from standard Gaussian. This defines the Markov chain as the probability of the variable $\boldsymbol{z}_t$ is only dependent on the previous variable $\boldsymbol{z}_{t-1}$ or equally:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \text{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\right] \quad \forall t \in \{2, \dots, T\} \tag{3}$$

As the t gets larger, the probability of $\mathbf{z}_t$ tends to a unit Gaussian. For the reverse diffusion model, there exits a closed-form expression for $q(\mathbf{z}_t|\mathbf{x})$ such that direct sampling of latent variable $\mathbf{z}_t$ is possible without calculating the intermediate steps, by the following relation:

$$q(\mathbf{z}_t|\mathbf{x}) = \text{Norm}_{\mathbf{z}_t}\left[\sqrt{\alpha_t} \cdot \mathbf{x}, (1-\alpha_t)\right] \tag{4}$$

Where α_t can be interpreted as the cumulative fraction of the data remaining in the latent variable with the relation $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$. Equation 4 is also known as the *diffusion kernel* which is a normal distribution with a mean approaching to zero, and variance approaching 1. Diffusion kernel tends to a standard Gaussian distribution as the value of t increases. The Equation 4 is crucial for efficiently sampling latent variables within our diffusion model framework, as it obviates the need for computing intermediary latent states. Therefore, this analytical simplification greatly enhances the training efficiency and speed of diffusion model algorithms.

As long as the noise schedule values are small, the assumption that reverse diffusion distributions follow a Gaussian is valid. This assumption then permits the employment of deep learning architectures to predict the Gaussian parameters. Specifically, neural networks are used to estimate the mean of the Gaussians that maps the latent variable, $\mathbf{z}t$, back to its preceding state, $\mathbf{z}t-1$. The direct maximization of the likelihood of observed data, achieved by marginalizing over the latent space, proves intractable. Therefore, Jensen's inequality is invoked to establish a lower bound for the log-likelihood function[5]. Although the derivation is mathematically sophisticated and elegant, it surpasses the scope intended for this coursework. Consequently, we present the simplified loss function for training DDPM, omitting scaling factors for brevity.

$$L[\phi_{1\dots T}] = \sum_{i=1}^{I}\sum_{t=1}^{T}\left\|g_t\left[z_{it}, \phi_t\right] - \varepsilon_{it}\right\|^2 = \sum_{i=1}^{I}\sum_{t=1}^{T}\left\|g_t\left[\sqrt{\alpha_t} \cdot x_i + \sqrt{1-\alpha_t} \cdot \varepsilon_{it}, \phi_t\right] - \varepsilon_{it}\right\|^2, \tag{5}$$

The loss function, in Equation 5, aggregates the squared error across all time steps T and data instances I. It compares the predicted noise, $\mathbf{g}_t$, against the actual noise, $\varepsilon_{\mathbf{it}}$, for each latent representation transformed by the diffusion kernel. The function $\mathbf{g}_t$, represented by a U-Net in our analysis, approximates the noise ε given the diffusion-processed latent variables. This process mirrors an orchestra, where the U-Net—akin to a diverse assembly of instruments—interprets the input data, translating it into a latent symphony. Meanwhile, the diffusion model, much like a conductor, harmonizes the ensemble's performance by meticulously introducing slight Gaussian noise according to a pre-determined schedule.

Furthermore, the practice of predicting the noise, ε, instead of predicting the posterior mean of the forward process was shown to be more effective by Ho et. al [3]. This parameterization of the loss function to predict the noise intrinsically obeys the Langevin dynamics while resembling denoising score matching. This distinction is what sets apart DDPM from classical diffusion models and improves the quality of the generated images.

The DDPM training involves two key phases: the forward pass, where data is noised and loss computed, and the sampling phase, where the model predicts prior latent variables to generate samples via the reverse process. Here's an overview of the forward pass:

Algorithm 1 Diffusion model training - Algorithm 18.1 of [5]

```
1:  Input: Training data $\mathbf{x}$
2:  Output: Model parameters $\phi_t$
3:  while not converged do                                              ▷ Repeat until converged
4:      for $i \in B$ do                                ▷ For every training example index in batch
5:          $t \sim \text{Uniform}[1, \ldots, T]$                              ▷ Sample random timestep
6:          $\varepsilon \sim \text{Norm}[0, I]$                                          ▷ Sample noise
7:          $\ell_i = \left\|\mathbf{g_t}\left[\sqrt{\alpha_t} \cdot \mathbf{x_i} + \sqrt{1-\alpha_t} \cdot \varepsilon, \phi_t\right] - \varepsilon\right\|^2$    ▷ Compute individual loss
8:      end for
9:      Accumulate losses for batch and take gradient step
10: end while
```

The process starts by randomly sampling a time step and corresponding Gaussian noise, utilizing these to compute the latent variable z_t. This variable is fed into the U-Net, predicting the step's specific noise, and culminates in a scalar loss for parameter optimization through back-propagation. Concurrently, the generative, reverse diffusion phase begins by drawing z_T from a presumed Gaussian. The algorithm iteratively refines back to $\hat{z}_{t-1}$, excluding the final step to enhance data quality, by avoiding additional noise injection at $z1$[5].

Algorithm 2 Sampling - Algorithm 18.2 of [5]

```
1: Input: Model, $g_t[\cdot, \phi_t]$
2: Output: Sample, $x$
3: $z_T \sim \text{Norm}_z[0, I]$                                             ▷ Sample last latent variable
4: for $t = T$ down to 2 do
5:     $\hat{z}_{t-1} = \frac{1}{\sqrt{1-\beta_t}} z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}} g_t[z_t, \phi_t]$    ▷ Predict previous latent variable
6:     $e \sim \text{Norm}_e[0, I]$                                                ▷ Draw new noise vector
7:     $z_{t-1} = \hat{z}_{t-1} + \sigma_t e$                          ▷ Add noise to previous latent variable
8: end for
9: $x = \frac{1}{\sqrt{1-\beta_1}} z_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}} g_1[z_1, \phi_1]$    ▷ Generate sample from z1 without noise
```

Incorporating these functions into a DDPM class facilitates real-time sampling post-training iterations, allowing visual progress assessment via generated images. Furthermore, time encoding, integral to varying noise schedules, ensures the model's explicit recognition of the temporal dimension.

1.0.2 1(b)

The DDPM algorithm, applied to the MNIST dataset, reveals the loss curves for training and validation samples as depicted in Figure 1. A diffusion time (T) of 1000 and a linear noise schedule ranging from

10^{-4} to 0.02 were employed. Utilizing a U-Net architecture, the model features a CNN-Block composed of four hidden layers with output channels (16, 32, 32, 16) and incorporates zero padding. Optimization was performed using the Adam optimizer with a learning rate of 0.0002.

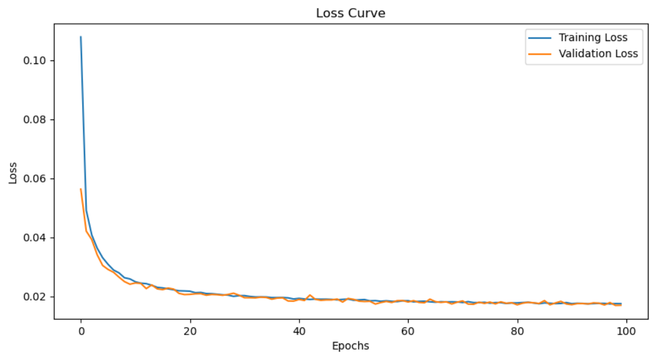


Figure 1: The training and validation loss curve for the DDPM model over 100 epochs.

Both the training and validation loss curves represent averages over a single epoch, with the training data comprising 468 batches of 128 samples each, and the validation data containing 78 batches of the same size. Notably, there's an initial sharp decline in loss values, suggesting the model quickly identifies a minimum. Interestingly, early on, the validation loss is lower than the training loss. This might seem unexpected since the model is trained on the training data, but it can be attributed to the timing of loss calculation: training loss is computed batch-wise within an epoch, while validation loss is calculated post-epoch, effectively making training loss appear higher in the early stages on the graph1 [6]. Additionally, the smaller size of the validation dataset could lead to higher variance in the validation loss, as seen in the graph's fluctuations. Ultimately, both loss curves stabilize, indicating diminished returns on further training beyond 50 epochs, chosen for efficiency despite potential insights from further epochs such as the double-descent phenomenon. This "vanilla" model is then evaluated against alternative hyperparameters, focusing on the noise schedule (β_t), which dictates the progression of Gaussian noise addition during diffusion. Contrary to the initial linear schedule, which increases linearly over diffusion time, geometric and custom schedules are tested to explore the effects of this parameter on image synthesis. Generally, the noise level is increased in later diffusion steps to transition smoothly to a Gaussian distribution. For the geometric noise schedule, the β_t values are calculated geometrically from the initial value β_1 to the final value β_2 using the formula $\beta_t = \beta_1 \left(\frac{\beta_2}{\beta_1}\right)^{\frac{t}{T}}$. The custom noise schedule introduces a variable rate of increase, initially accelerating and then decelerating towards the final steps, inspired by cosine annealing[4], and is defined as: $\beta_t = \frac{\beta_2 - \beta_1}{2}(1 - \cos(\pi t)) + \beta_1$. This approach aims to provide a flexible introduction of noise into the latent space during the forward diffusion process. Figure 2 visually compares the progression of β_t values across diffusion time t for the different noise schedules, illustrating the strategic impact of these variations on the model's functionality.

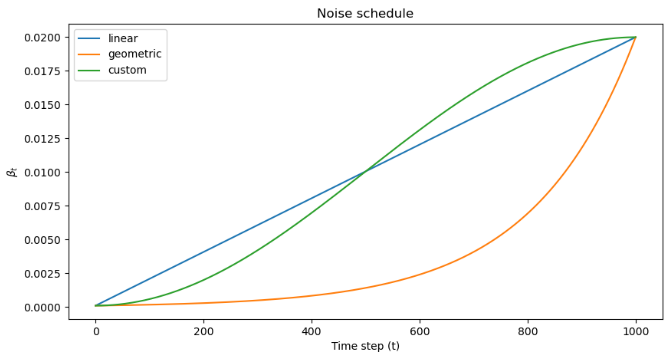


Figure 2: The plot demonstrates the progression of noise schedule values across diffusion time for different noise schedules.

The training and validation loss curves across 50 epochs under different noise schedules are depicted in Figure 3. The observation of higher training losses in initial epochs, attributed to a half-epoch phase shift, holds true for all schedules. Among the schedules, the linear noise schedule achieves the lowest loss values, followed by the custom and geometric schedules, with the geometric schedule recording the highest loss. It's crucial to note that this analysis, limited to 50 epochs due to practicality, suggests that a linear noise schedule optimally reduces the mean squared error for both training and validation datasets. However, a more extensive training period might reveal different dynamics and efficiencies of each noise schedule.

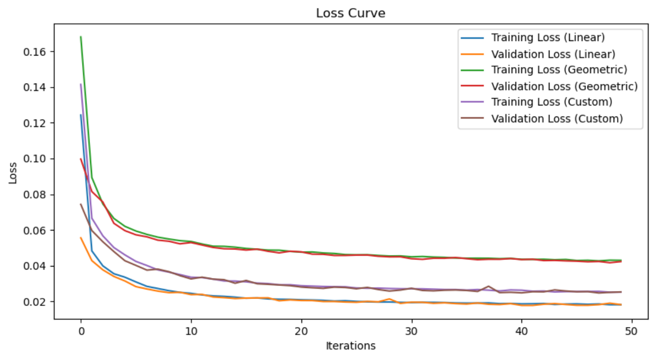


Figure 3: The training and validation loss curve for different noise scheduler over 50 epochs.

For evaluating generated images from the model, two metrics were used due to the MNIST dataset's unique challenges. The first, Peak Signal-to-Noise Ratio (PSNR), measures image quality by comparing the highest possible signal value against the mean squared error (MSE) between a generated and a true image. Higher PSNR indicates better quality, implying lower MSE when adjusted for the image's maximum signal. The PSNR is calculated with $20 \cdot \log_{10}(\frac{MAX}{\sqrt{MSE}})$. PSNR scores for each noise schedule are plotted in Figure 4. For analysis, each generated image was assessed against 100 validation images, with PSNR scores averaged to gauge overall synthesis quality. This method aims to reflect the model's effectiveness in generating high-fidelity images by comparing them to a broad set of true digits.

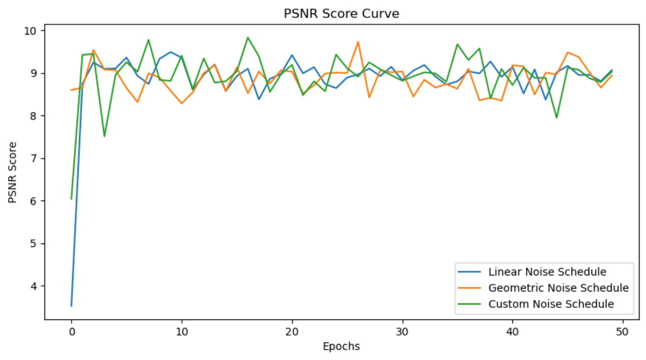


Figure 4: The plot of PSNR scores averaged over each epoch for different noise schedules.

PSNR offers a straightforward approach for grayscale images by requiring only a single-channel MSE calculation. However, it doesn't compare images directly but averages scores across a set, potentially matching dissimilar images. The observed PSNR around $9dB$ across noise schedules suggests a baseline quality level. Additionally, the Structural Similarity Index Measure (SSIM) evaluates perceptual quality through luminance, structure, and contrast comparisons, with values from -1 to 1; higher scores denote greater similarity. Like PSNR, SSIM scores are averaged over 100 validation images per training step, aiming to gauge the resemblance of generated samples to actual digits, providing insight into the model's effectiveness.

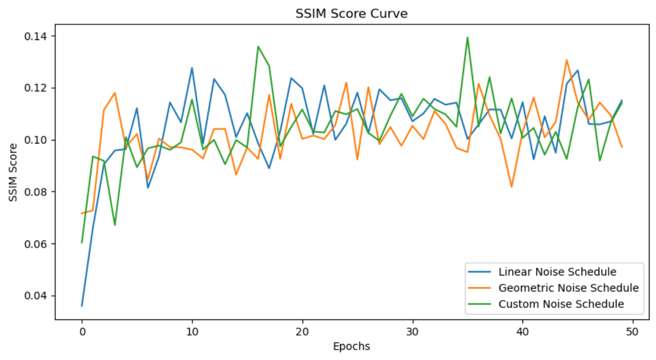


Figure 5: The plot of SSIM scores averaged over each epoch for different noise schedules.

SSIM scores, oscillating between 0.8 to 0.14 for all noise schedules, suggest minimal similarity between generated samples and 100 ground truth images, possibly indicating that pixel value proximity doesn't always correlate with higher quality. The averaging process could dilute the scores, especially if the exact matching digit is sparse or absent in the compared samples. This outcome points towards the necessity for a deeper analysis that leverages high-level image features. Consequently, the discussion will next turn to a more sophisticated, feature-based evaluation metric. However, before introducing this metric, the impact of network capacity on PSNR and SSIM scores is examined.

Expanding the network capacity from the initial configuration $(16, 32, 32, 16)$ to $(64, 128, 256, 128, 64)$ aims to enhance the model's ability to capture the latent space's semantic features. Comparative analysis of training and validation loss for this augmented network capacity versus the standard model will shed light on whether increased complexity translates to performance improvements, as illustrated in subsequent loss curve comparisons in Figure 6.

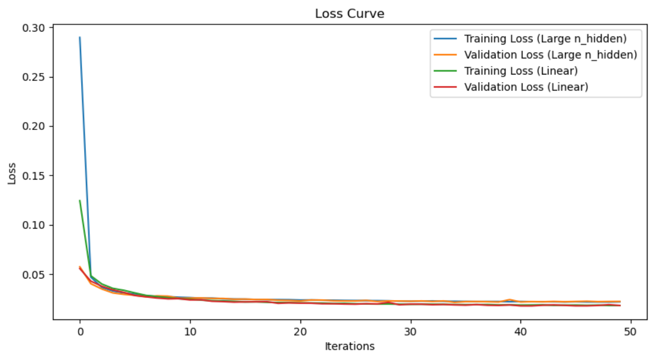


Figure 6: The training and validation loss curves for the default model and the model with expanded U-Net capacity.

The SSIM and PSNR scores comparison is given in Figure 7.

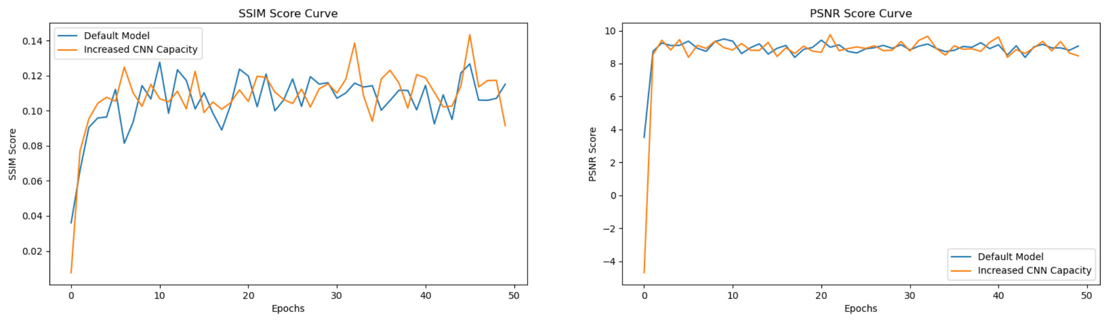


Figure 7: SSIM and PSNR scores averaged over each epoch for model with increased capacity compared to the default model.

Observations from Figure 7 reveal fluctuations in both SSIM and PSNR scores for the models. Notably, the model with expanded capacity initially registers negative PSNR values, suggesting dominant noise over signal in early epochs. Both configurations demonstrate parallel improvements, with progress plateauing after initial epochs, mirroring the trends seen in their loss curves. However, the inherent limitation of pixel-based metrics like SSIM and PSNR is their sensitivity to transformations without accounting for semantic features such as shapes or edges - critical for evaluating generated digit image quality. This highlights the need for metrics that consider these higher-level features to more accurately assess image synthesis performance.

1.0.3 1(c)

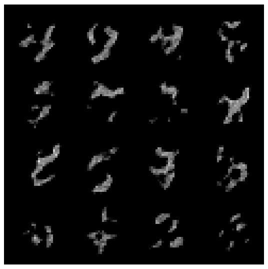
For a deeper evaluation of image synthesis, the Freschet Inception Distance (FID) serves as a sophisticated metric, focusing on semantic and perceptual details like textures, shapes, and edges, which is especially pertinent for the MNIST dataset. However, FID, while being a benchmark for assessing generative model performance, has limitations. It hinges on comparing feature vectors from real and generated images, derived using an Inception v3 model primarily trained on natural, color images, not digits or grayscale[2]. This necessitates adapting grayscale MNIST images to a three-channel format to align with the model's RGB input, raising questions about FID's effectiveness for grayscale image evaluation. Moreover, FID's focus isn't on the classification accuracy of generated images against Inception v3's training objects but rather on analyzing the feature vectors from the model's penultimate pooling layer. Here, the mean and covariance of these vectors quantify the distance in feature space between real and synthesized distributions, with lower scores indicating closer resemblance to real images, thus implying higher synthesis quality. The results of this analysis, gathered in FID scores, offer insight into the comparative quality of images generated under various noise schedules given in Table **??**.

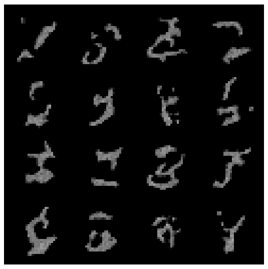
Table 1: FID Scores for Different Noise Schedules and Capacities

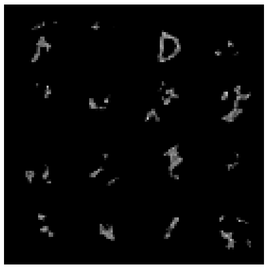
	Linear Noise Schedule	Geometric Noise Schedule	Custom Noise Schedule	Increased CNN Capacity
FID Score	50.02	218.77	58.42	24.54

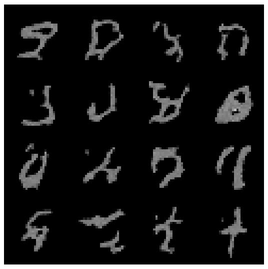
For an in-depth quality assessment of synthesized images, FID scores were computed for images generated by fully trained models over 50 epochs, using 1,000 generated images compared to 1,000 authentic images. Consistent with the training loss trends, these FID scores reveal that the linear noise schedule produced the highest-quality samples, followed by the custom and geometric schedules. Moreover, comparing the standard model under a linear noise schedule to an enhanced capacity model using the same schedule underscores a marked improvement in FID score for the latter. This improvement was anticipated, as the increased capacity model had a greater ability to capture digit structural features.

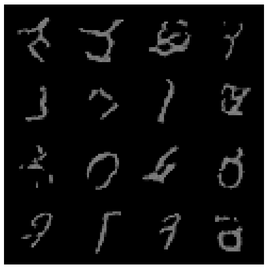
Illustrative comparisons between high and low-quality samples from each model type are showcased in Figure 8, with low-quality samples drawn after 6 epochs and high-quality ones after 49 epochs. This visual representation further elucidates the progression and efficacy of each training strategy.

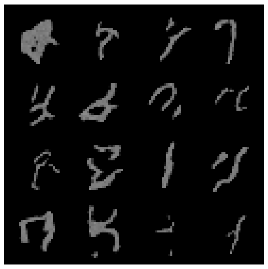

Linear Schedule
Low-Quality

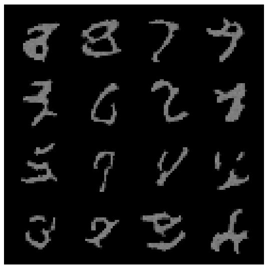

Geometric Schedule
Low-Quality


Custom Schedule
Low-Quality


Increased Capacity
Low-Quality


Linear Schedule
High-Quality


Geometric Schedule
High-Quality


Custom Schedule
High-Quality

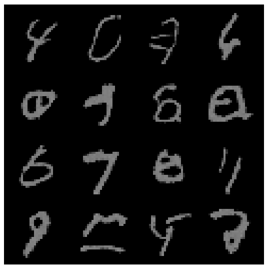

Increased Capacity
High-Quality

Figure 8: High and low-quality samples generated for all 4 models: linear schedule, geometric schedule, custom schedule, and increased capacity with a linear schedule. Low-quality samples are generated after 6 epochs of training whereas high-quality samples are generated after 49 epochs of training.

The observation that the custom noise schedule's low-quality samples appear more distorted than those from linear and geometric schedules is noteworthy. Yet, high-quality samples from the custom schedule showcase clearer digit representations, notably for digits 8, 7, 6, and 2. This can be attributed to the custom schedule's slower rate of noise increase in later diffusion steps, as illustrated in Figure 2, potentially enhancing sample quality in the concluding training phases. Additionally, the network capacity's influence is evident in both high and low-quality sample comparisons. The enhanced capacity model yields more recognizably digit-like images, particularly in high-quality samples displaying digits 4, 0, 6, 7, and 9, agreeing with the lowest FID scores.

2 Custom Degradation

2.0.1 2(a)

The proposed Diffusion with Entropy Scale (DES) introduces a novel custom degradation strategy for DDPMs, diverging from the traditional approach of adding Gaussian noise with constant variance. Contrary to deterministic methods explored in studies like Cold Diffusion[1], DES combines stochastic noise sampling with dynamic, content-dependent entropy scaling. This methodology entails calculating entropy across batches of images by flattening their pixel values into a histogram. With 256 bins spanning the range between the highest and lowest pixel values in the batch, the histogram is normalized to function as a probability distribution. Shannon's entropy formula is applied:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i), \tag{6}$$

where $P(x_i)$ denotes the probability of pixel value x_i. This approach yields a robust measure of entropy, reflecting the variability in the batch's pixel distribution. The resulting entropy is then mapped through a sigmoid function to scale the stochastic noise ε in a content-aware manner. Batches with higher entropy undergo less noise reduction, enabling a nuanced approach to noise addition. This dynamic method stands in contrast to predetermined noise schedules, allowing noise variance to adapt based on batch entropy, thus reflecting the complexity of the current content.

Furthermore, DES's adaptive noise scaling encourages the model, especially the U-Net architecture, to discern and prioritize learning features of varying entropy levels. This aligns the degradation process more closely with natural image degradation patterns, offering a sophisticated approach to modifying the diffusion process. By adjusting noise addition in line with the intrinsic content complexity of images, DES could potentially enhance model performance.

2.0.2 2(b)

The training and validation loss curves for the DES model, trained on the MNIST dataset over 50 epochs, are illustrated in Figure 9. Initially, the training loss curve slightly lags due to being computed mid-epoch, showing both curves trending downward and stabilizing after the 15th epoch.

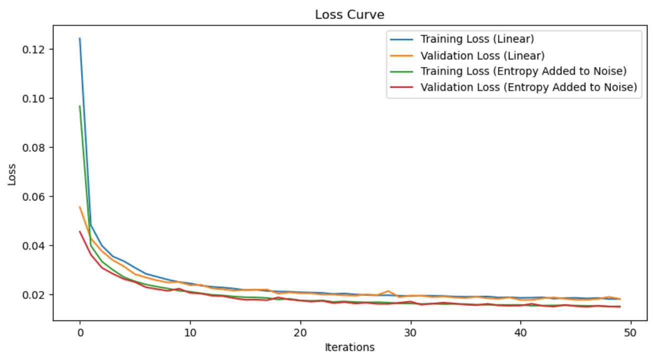
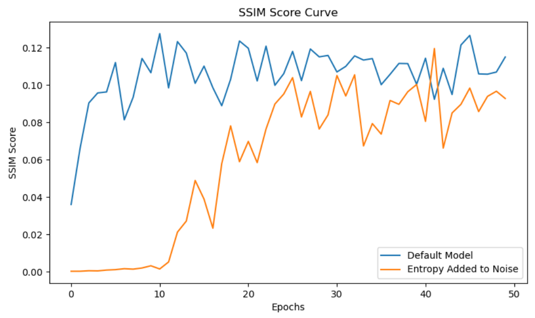


Figure 9: The training and validation loss curves for the DES model compared to the default DDPM model over 50 epochs.

Regarding SSIM and PSNR metrics shown in Figure 10, the DES method initially records lower scores than the standard model, indicating lower initial sample quality. However, these metrics alone may not fully capture the nuanced improvements made by incorporating entropy scaling. Thus, a deeper analysis utilizing the Frechet Inception Distance (FID) score, alongside visual comparisons of generated samples, will provide a more comprehensive assessment of the DES model's effectiveness compared to the baseline model.



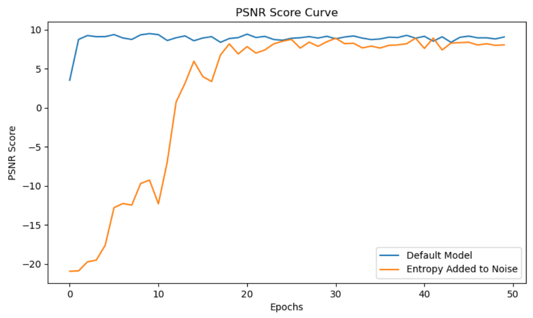


Figure 10: SSIM and PSNR scores averaged over each epoch for the DDPM and DES models.

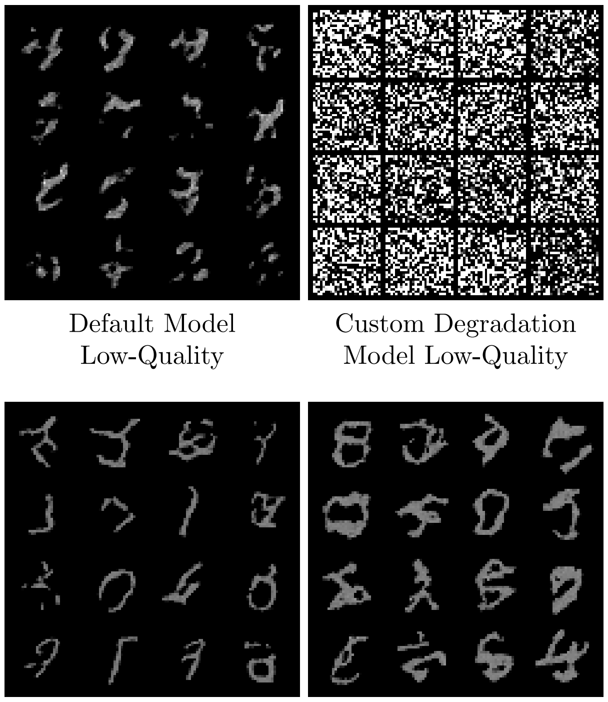
2.0.3 2(c)

The FID score comparison for the default model and the custom model is given in Table 2. The FID is calculated between $1,000$ generated samples for each model and the corresponding $1,000$ actual images. The custom degradation model has a lower FID for the final model indicating better quality sample generation.

Table 2: FID Scores for Default DDPM and Custom DES

	Default Model	Custom Degradation
FID Score	50.02	47.80

Examining the generated samples from early (6 epochs) and later (49 epochs) training stages in Figure 11, early DES model outputs are less distinguishable, aligning with initially lower PSNR and SSIM scores. However, by the end of training, the high-quality samples from DES demonstrate noticeable improvements, with digits like 8, 4, and 0 becoming discernible, showcasing the model's capacity to generate comparable quality samples relative to the default approach.



Default Model Low-Quality

Custom Degradation Model Low-Quality

Default Model High-Quality

Custom Degradation Model High-Quality

Figure 11: Comparison of high and low-quality samples for the default model and custom degradation model.

One key distinction between high-quality samples of the models is that the width of the digits seems to be thicker in the entropy-scaled error method. This might be due to the inclusion of batch entropy instead of single-image entropy. While changing this aspect of entropy calculation might improve the results, binning and creating histograms for each pixel is more computationally heavy for each image compared to a batch of images. Given the computational trade-offs, it remains challenging to definitively ascertain which model yields superior digit images based on the limited 50-epoch evaluation. Nonetheless, it's noteworthy that early in training, the default model produces more recognizable digit samples compared to the essentially white-noise-like output of the custom degradation model. The quality of the DES model's samples notably increases after approximately 20 epochs, as evidenced by the performance metrics presented. This improvement trajectory underscores the dynamic nature of the DES model's learning process and its potential for further exploration and optimization.

References

[1] A. Bansal, E. Borgnia, H.-M. Chu, J. S. Li, H. Kazemi, F. Huang, M. Goldblum, J. Geiping, and T. Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise, 2022.

[2] J. Brownlee. How to implement the frechet inception distance (fid) for evaluating gans, October 2019.

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

[4] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[5] S. J. Prince. *Understanding Deep Learning.* MIT Press, 2023.

[6] A. Rosebrock. Why is my validation loss lower than my training loss?, October 2019.

A Generative AI

ChatGPT 3.5 was used for suggesting alternative wordings, grammar suggestions, and proofreading while writing the report. The following prompts were used during this process

- ”How can I write this equation in LaTeX format?” *[input equation]*
- ”Provide BibTex citation format for this website *[input website]*”
- ”How to represent/write *[input symbol]* in LaTeX?”
- ”What is another word for *[input]* word?”
- ”Is this sentence grammatically correct? *[input sentence]*”
- ”Is this paragraph clear for a reader? *[input paragraph]*”
- ”How to rephrase this sentence to make it more clear? *[input sentence]*”

For the report writing, the outputs were partially adapted in a way that only alternative wordings were used and not the whole output while rephrasing the discussion parts of the report. The LaTex code for equations was adapted from the suggestion. The suggestions for BibTeX citations are also adapted.

Furthermore, the suggestions from the autocomplete feature of ‘GitHub Copilot‘ were utilized during the documentation of the software, and code development of the project such as adjusting the style of the plots generated.