MPhil Data Intensive Science
University of Cambridge

Application of Machine Learning Coursework
Cold Zoom Diffusion

Andreas Vrikkis

March 28 , 2024
LaTeX Word count: 2875

Abstract

Diffusion models in image generation are based on a strategy that degrades images, coupled with a reconstruction operation that reverses this degradation to generate samples from noise. In this work, we design a novel deterministic degradation strategy named Cold Zoom Diffusion, which degrades images by increasingly zooming into their central pixels. We compare this model to a Denoising Diffusion Probabilistic Model (DDPM) with linear and cosine noise schedules. All models are trained on the MNIST dataset and their samples are evaluated using the Fréchet Inception Distance (FID) metric. The Cold Zoom Diffusion model performs comparably to the linear DDPM model despite its deterministic nature and small reconstructor network. However, it demonstrates a bias towards certain digits, attributed to the deterministic nature of its degradation process, which lacks noise addition in the reconstruction phase.

Contents

1 Introduction

Diffusion models form a class of deep generative models which has recently pushed foward the landscape of image generation. While Diffusion models exist in various forms, they all centre arount the idea of reversing the addition of random noise to an image. A network is trained to predict the noise added to an image at each step of the diffusion process, and at the reverse process, the network is fed with random noise to be reversed back to a new sample. Recently, the authors of ref.[1] demonstrate that the generative behaviour of diffusion models is not limited by the choice of random degradation, and that even completely deterministic degradation strategies can be used to generate high quality samples. In this work, we develop our own deterministic degradation strategy, Cold Zoom Diffusion, which degrades images by simply zooming into the central pixels of the image. We first train a provided Denoising Diffusion Probabilistic Model (DDPM) with different noise schedules, and then present the methodology and results of our Cold Zoom Diffusion model.

2 Denoising Diffusion Probabilistic Models

2.1 Background

This section outlines the provided Denoising Diffusion Probabilistic Model (DDPM) and its training algorithm. DDPM operates as a parameterised Markov chain, executing a series of diffusion steps that incrementally introduce random noise into data. Its training objective is to reverse this diffusion process, generating data samples from pure noise (noise seeds). In the forward process, the model takes a data sample $\mathbf{x}$ and maps it through a series of intermediate latent variables $\mathbf{z}_{1...}\mathbf{z}_T$, formulated as:

$$\mathbf{z}_1 = \sqrt{1-\beta_1} \cdot \mathbf{x} + \sqrt{\beta_1} \cdot \boldsymbol{\epsilon}_1 \tag{1}$$

$$\mathbf{z}_t = \sqrt{1-\beta_t} \cdot \mathbf{z}_{t-1} + \sqrt{\beta_t} \cdot \boldsymbol{\epsilon}_t \quad \text{for} \quad t = 2, ..., T \tag{2}$$

Here, $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \boldsymbol{I})$ is noise drawn from a standard normal distribution, and β_t denotes the noise schedule—a series of hyperparameters determining the noise level at each diffusion step. It can be shown [2] that we can calculate $\mathbf{z}_T$ directly form $\mathbf{x}$ using:

$$\mathbf{z}_T = \sqrt{\alpha_t} \cdot \mathbf{x} + \sqrt{1-\alpha_t} \cdot \boldsymbol{\epsilon} \tag{3}$$

where $\alpha_t = \prod_{i=1}^{t}(1-\beta_i)$. Hence the intermediate steps are not needed in the training process. As the number of time steps increases, the latent state converges towards standard Gaussian noise.

To reverse the diffusion process, a neural network is trained to estimate the noise $\boldsymbol{\epsilon}_t$ added to a training sample $\mathbf{x}$ at each time step t. This is achieved by randomly selecting a time step t, calculating the corresponding latent state $\mathbf{z}_t$, and then employing a loss function to guide the model in predicting the added noise $\boldsymbol{\epsilon}_t$. We will refer to this model as the decoder network $\mathbf{g}_t$. To sample from the model, we start with a standard Gaussian noise sample $\mathbf{z}_t \sim \mathcal{N}(0, \boldsymbol{I})$ which represents a fully degraded image at time step $t = T$. In the reverse process, we can recast eq. 2 into:

$$\mathbf{z}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\boldsymbol{\epsilon}_t. \tag{4}$$

We can use this form to define the reverse process, by approximating the previous latent state $\mathbf{z}_{t-1}$ from the current latent state $\mathbf{z}_t$ using the noise prediction of the decoder network $\mathbf{g}_t$.

Starting from the completely degraded image $\mathbf{z}_T$, we loop over the time steps in reverse order, using:

$$\hat{\mathbf{z}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t\left[\mathbf{z}_t, \phi_t\right] \tag{5}$$

where ϕ_t is the network's parameters at time step t. In each time step from $t = T \ldots 2$ we also add Gaussian noise to the latent state to allow for stochasticity in the sampling process and to explore the space of possible images.

2.2 Provided Model Architecture

In the provided model architecture, the decoder $\mathbf{g}_t$ which predicts the noise is built using a convolutional neural network (CNN). The CNN consists of a series of convolutional layers followed by a Layer Normalisation (LayerNorm) and GeLU activation functions. The LayerNorm normalises the feature vector based on its own mean and standard deviation, while the GeLU activation function is a smoother approximation of the ReLU function. The final convolutional layer adjusts the output dimensions to match the original number of input channels. As the noise added is dependent on the time step t, the model must integrate this temporal information into its predictions. It employs sinusoidal embeddings to transform scalar time values into vector representations. The code generates multiple frequencies, applying sine and cosine functions to the time value t, modified by these frequencies. These results are then processed through linear layers to create a high-dimensional representation of the time step. The network first processes $\mathbf{x}$ through an initial convolutional layer. It then adds time-encoded information to this output by broadcasting the time embeddings to the spatial dimensions of the CNNBlock's output. The subsequent convolutional layers further process this combined data. The final output is a tensor that incorporates both spatial and temporal information.

The training process consists of looping over the training data in batches and randomly sampling a time step t for each sample. The model then calculates the latent state $\mathbf{z}_t$ and uses the decoder network to predict the noise $\boldsymbol{\epsilon}_t$. The loss function used to train the model is the mean squared error (MSE) between the predicted noise and the actual noise. For a single sample, the loss is given by:

$$l_i = \|\mathbf{g}_t\left[\mathbf{z}_t, \frac{t}{T}|\phi_t\right] - \boldsymbol{\epsilon}_t\|^2 \tag{6}$$

The losses are accumulated for the batch and a gradient step is taken to update the model parameters. The model is trained using the Adam optimiser with a learning rate of 2^{-4} and batchsize of 128.

2.3 Noise Schedules and Training

The noise schedule β_t of the provided DDPM model is set to be a sequence of linearly increasing constants from $\beta_1 = 10^{-5}$ to $\beta_T = 0.02$. These values are small relative to the normalised MNIST pixel values ranging between [-0.5, 0.5]. Following ref.[3], we introduce a cosine noise schedule defined as:

$$f(t) = \cos\left(\frac{t/T+s}{1+s}\cdot\frac{\pi}{2}\right)^2 \tag{7}$$

$$\alpha_t = \frac{f(t)}{f(0)} \tag{8}$$

$$\beta_t = 1 - \frac{\alpha_t}{\alpha_t - 1} \tag{9}$$

where a small offset s is used to prevent β_t from becoming too small near $t = 0$. We also clip the values of β_t to the maximum value of 0.02, a limit chosen to align with the original noise schedule and to prevent singularities as t approaches T. With an $s = 0.008$ as in the original paper, we compare the cosine and linear noise schedules in Fig. 1. The cosine schedule is designed to add less noise in the intermeditate steps than the linear schedule, which allows for more information to be retained. The extremes at $t = 0$ and $t = T$ are mostly the same for both schedules. Fig. 2 displays the intermediate steps on an example image for each noise schedule. At $t = 300$, the linear schedule's latent images are almost entirely random, while the cosine schedule retains more information at the same time step. Fewer time steps are wasted during training and the cosine model is expected to perform better with the same number of time steps and training epochs.

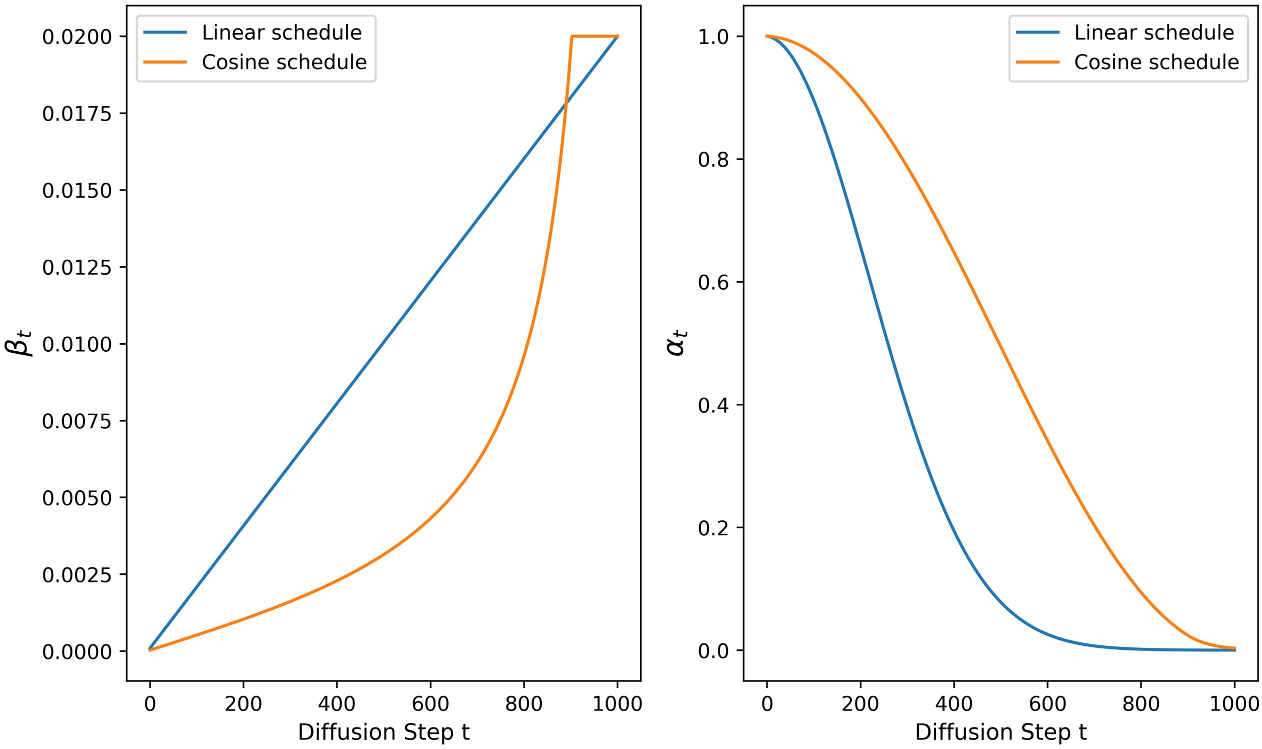

Figure 1: Linear and cosine noise schedules used in the DDPM model.

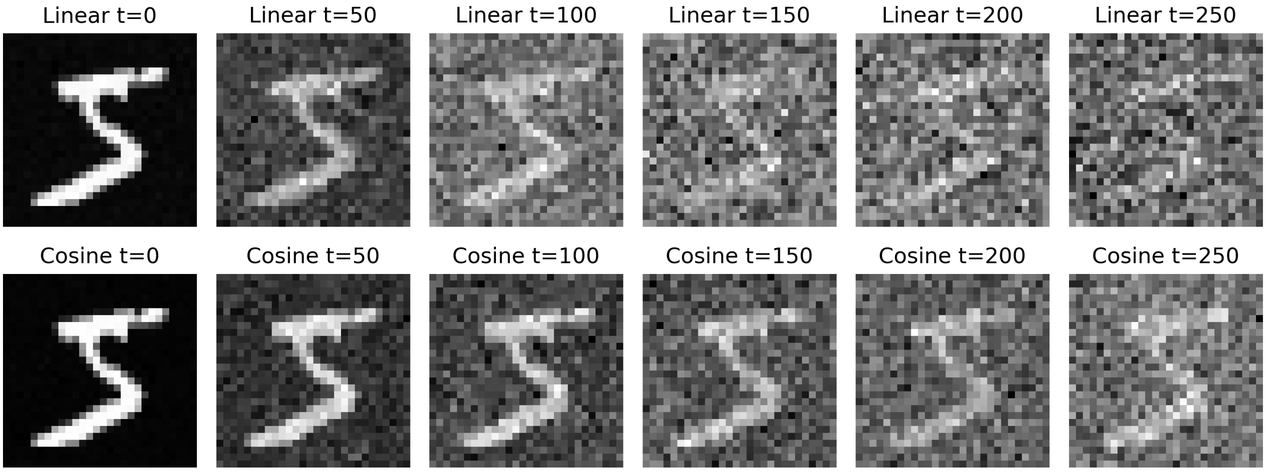

Figure 2: Latent images from linear (top) and cosine (bottom) noise schedules at linearly spaced time steps. The cosine schedule adds noise more gradually than the linear schedule.

We train both models the cosine and linear noise schedule DDPMs for 100 epochs, a learning rate of 2^{-4} and a batch size of 128. The decoder $\mathbf{g}_t$ consists of 4 intermediate CNN blocks of channel sizes [16,32,32,16].

Before training, we split the MNIST dataset into training and test sets. During training, we record the mean loss across all batches for that particular epoch $\overline{L}_{MSE}$. We then set the model to evaluation mode and sample from the model using the reverse process, while also recording the average test loss. The loss curves for the linear and cosine schedule DDPM models are shown in Fig.3. In both the training and test sets, the cosine schedule model reaches a lower loss than the linear schedule model. This reinforces the view that the model trains more efficiently and that more time steps are meaningful in the training process for the cosine schedule.

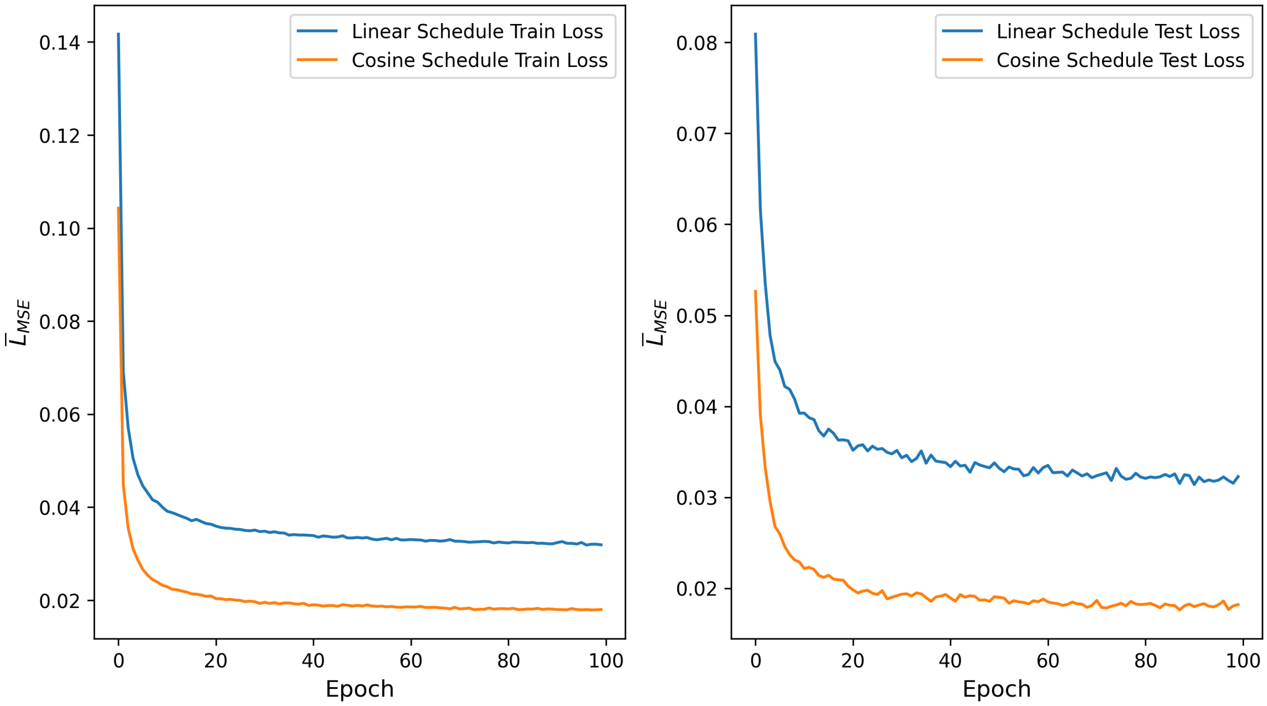

Figure 3: Mean squared error loss curves for the linear and cosine noise schedule DDPM models.

2.4 Sample Quality

To assess the quality of the samples generated by the models, we use the Fréchet Inception Distance (FID) metric [4] from `torchmetrics.image.fid` [5]. FID is used to measure the discrepancy between two image sets. In our case, we compare a set of 500 generated samples from our DDPM models with respect to the held-out MNIST test-set. To calculate FID, the image sets are passed through a pre-trained InceptionV3 network trained on the ImageNet classification task [6]. The InceptionV3's embeddings are then used to calculate the FID between the generated and real image sets, with lower values indicating better sample quality. However, it is important to acknowledge FID's limitations, especially as outlined in [7]. It is important to note that while FID is widely used for image generation evaluations, it has significant limitations [7]. Another limitation is that this metric is designed for natural, RGB images and may not be as informative for simpler datasets like MNIST.

The FID scores for the linear and cosine schedule models are shown in Fig.4. The cosine schedule achieves a lower FID score than the linear schedule model, indicating that the samples generated by the cosine schedule model are closer to the real MNIST images. This is consistent with the lower loss values achieved by the cosine schedule model during training. The downward trend in FID scores for both models implies potential improvements with extended training.

Fig.5 and Fig.6 display high and low quality samples from the linear and cosine schedule models, respectively. It is generally difficult to visually distinguish between the samples generated by the two models. Both models generate samples that resemble the MNIST dataset, but the cosine schedule tends to generate darker backgrounds surrounding the digits. Both models sometimes generate symbols that are not digits. The difference in FID scores between the two models is likely due to the cosine schedule model generating more realistic samples on average, rather than generating more high quality samples generally.

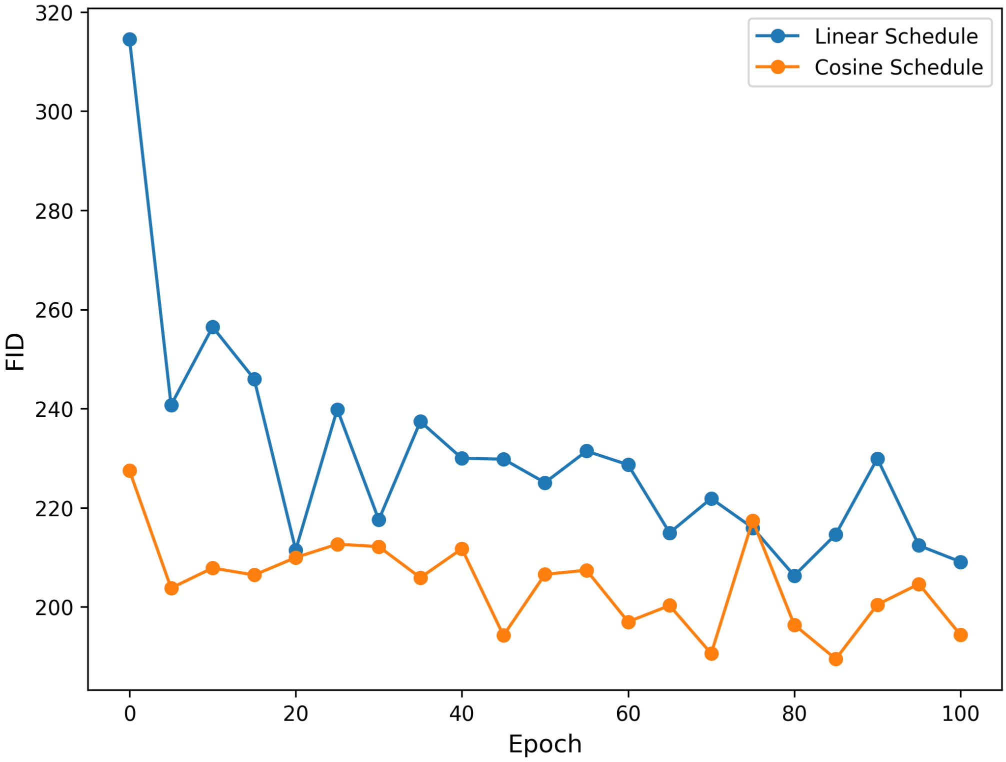

Figure 4: FID scores for the linear and cosine noise schedule DDPM models. The cosine schedule model achieves a lower FID score, indicating that the samples generated are closer to the real MNIST images. This is consistent through almost all epochs, which implies it trains more efficiently.

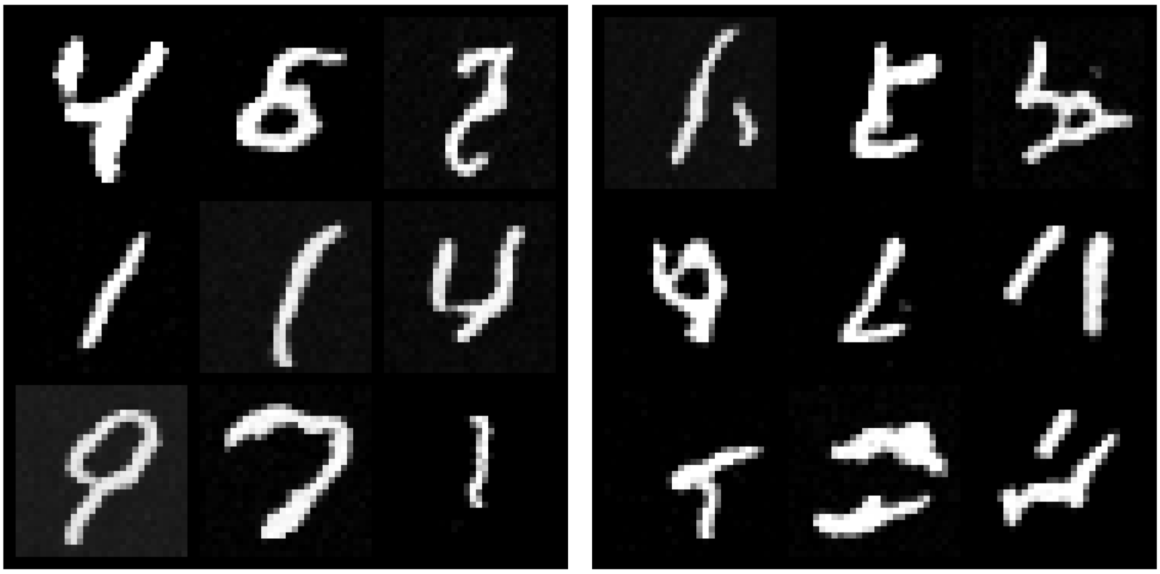

Figure 5: High quality samples (left grid) and low quality samples (right grid) generated by the linear schedule DDPM model. The pixel values are shifted to the range [0,1] for visualisation.

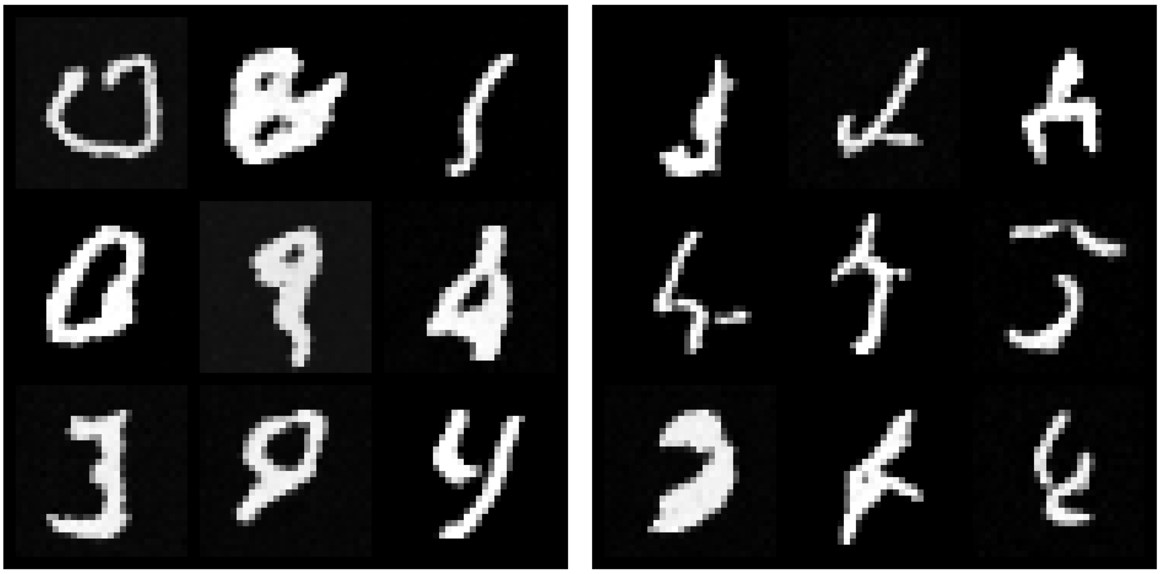

Figure 6: High quality samples (left grid) and low quality samples (right grid) generated by the cosine schedule DDPM model. The pixel values are shifted to the range [0,1] for visualisation.

3 Custom Degradation: Cold Zoom Diffusion

3.1 Cold Diffusion Background

Standard diffusion models consist of two components: an image degradation operator that adds Gaussian noise to the image, and a restoration operator approximated by a neural network to reverse the degradation process. In the context of cold diffusion, the degradation step is completely deterministic. Given an image $\mathbf{x} \in \mathbb{R}^N$ we can define a degradation of $\mathbf{x}$ using an operator D with severity t as $\mathbf{z}_t = D(\mathbf{x}, \frac{t}{T})$ where T is the maximum severity. The operator needs to satisfy $D(\mathbf{x}, 0) = \mathbf{x}$. We also need a restoration operator R that can reverse the degradation process. and satisfies $R(\mathbf{z}_t, t) \approx \mathbf{x}$. Practically, this is achieved through a reconstructor network $\mathbf{g}_t$, which aims to reconstruct the original image. The reconstruction can be expressed as:

$$\mathbf{g}_t\left[\mathbf{z}_t, \frac{t}{T}|\boldsymbol{\phi}_t\right] = \hat{\mathbf{x}} \approx \mathbf{x} \tag{10}$$

where $\boldsymbol{\phi}_t$ are the neural network parameters that are optimised during training.

Since we are predicting the original image, the latent degradation image $\mathbf{z}_{t...T}$ must contain information about the distribution of the original images. The restoration network is trained by minimising the loss function between the predicted original image and the actual image with respect to ϕ, given by:

$$L_\phi = \|\mathbf{g}_t\left[\mathbf{z}_t, \frac{t}{T}|\boldsymbol{\phi}_t\right] - \mathbf{x}\|^2 \tag{11}$$

Note the difference between the DDPM model's loss in eq.(6), where the loss is calculated between the predicted noise and the actual noise. In contrast, the restoration operator in cold diffusion is specifically trained to reconstruct the original image prior to degradation, rather than predicting the noise added.

3.2 Cold Zoom Diffusion

We introduce a deterministic degradation strategy that centres around zooming into the central pixels of an image, progressively discarding peripheral information at each step. We call this strategy Cold Zoom Diffusion. For a given image $\mathbf{x}$ and time step t, the degradation operator $\mathcal{Z}(\mathbf{x}, t)$ crops the image into its central pixels before resizing it back to its original dimensions. Considering MNIST images are 28×28 pixels, we define the zooming operation as alternate cropping from the top-left and bottom-right corners. For instance, for $t = 1$, the operator crops the image to 27×27 pixels by discarding the top-most row and left-most column, and then upscales it back to 28x28 using nearest-neighbour interpolation. Thus, at time step t, the image is cropped to the central $(28 - t) \times (28 - t)$ pixels and resized back to 28×28. The final cropping resolution is set at the central 4×4 pixels, leading to a maximum time step $T = 24$. We choose this T because the central 4×4 pixels contain minimal information about the original image, but enough to allow for a reconstruction prediction. To implement $\mathcal{Z}$, we use `torchvision.transforms.functional.resized_crop` [8], specifying the location of the top-left edge and the cropping dimensions. The interpolation mode is set to `InterpolationMode.NEAREST`, so each pixel in the resized image is assigned the value of the nearest pixel from the original image [9]. Fig.7 shows the zooming progression for an example image at different time steps.

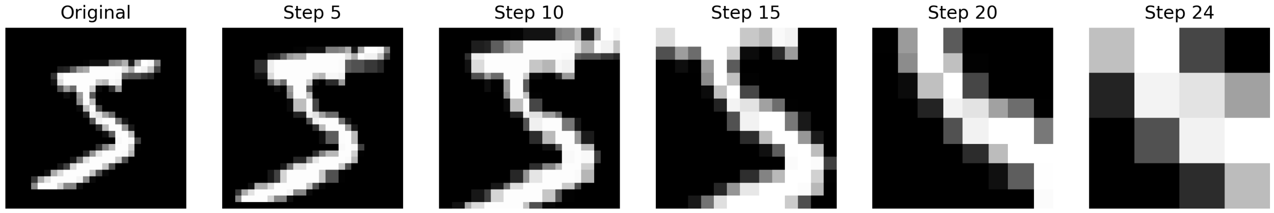

Figure 7: Zooming progression for a sample MNIST image at different time steps. The image is cropped by alternating between cropping the top-left and bottom-right corner dimensions. The fully degraded image at $t = 24$ is a nearest-neighbour resize of the central 4×4 pixels. The pixel values are shifted to the range [0,1] for visualisation.

The reconstruction operator is implemented with the same CNN-based network $\mathbf{g}_t$ described in section 2.2. The loss is implemented using MSE loss and the training process is provided in Algorithm 1.

Algorithm 1 Cold Zoom Diffusion Training

```
Input: Training data $\mathbf{x}$, Number of epochs N
for epoch = 1 to N do
    for $i \in \mathcal{B}$ do                                  ▷ For every training example index in batch
        $t \sim \text{Uniform}\,[1, 2, \ldots, 24]$                      ▷ Sample random timestep from 1 to 24
        $\mathbf{z}_t = \mathcal{Z}(\mathbf{x}_i, t)$                            ▷ Apply degradation zoom operator
        $l_i = \|\mathbf{g}_t\left[\mathbf{z}_t, \frac{t}{24}|\boldsymbol{\phi}_t\right] - \mathbf{x}\|^2$              ▷ Compute individual MSE loss
    end for
    Accumulate losses for batch and take gradient step
end for
```

Post-training, we employ the same operators sequentially to reverse fully degraded seeds and generate samples. In Cold Zoom Diffusion, a fully degraded latent $\mathbf{z}_T$ is a 28×28 image that was resized from a central 4×4 pixel region. Therefore, we require a method to generate these 4×4 central pixels, representative of the MNIST data distribution, to use as starting point seeds $\tilde{\mathbf{z}}_T$ for sampling. We begin by cropping the central 4×4 pixels from all the MNIST images using $\mathcal{Z}(\mathbf{x}, t = 24)$ and recording each pixel value. The distribution of these central pixels is shown in a histogram in Fig.8. Using `torch.distributions`, we create a categorical distribution based on these 256 possible pixel values. We then sample 16 independent values from this distribution to form a 4 × 4 image. This image is upscaled to 28 × 28 via nearest-neighbour interpolation using `torch.nn.functional.interpolate()`. These upscaled images serve as the initial seeds $\tilde{\mathbf{z}}_T$, which are processed iteratively through a series of restoration and degradation steps to reconstruct MNIST image samples. The sampling process is outlined in Algorithm 2, adapted from the approach proposed by the authors in ref.[1].

```
Algorithm 2 Cold Zoom Diffusion Sampling
  Input: A generated starting seed $\tilde{\mathbf{z}}_T$          ▷ Sampled from central pixel distribution
  for $s = T, T-1, \ldots, 1$ do
      $\hat{\mathbf{x}} = \mathbf{g}_s \left[\tilde{\mathbf{z}}_s, \frac{s}{24} | \phi_s\right]$          ▷ Predict the original image
      $\tilde{\mathbf{z}}_{s-1} = \mathcal{Z}(\hat{\mathbf{x}}, s-1)$          ▷ Apply Degradation Zoom operator
  end for
  Return: Reconstructed image $\hat{\mathbf{x}}$
```

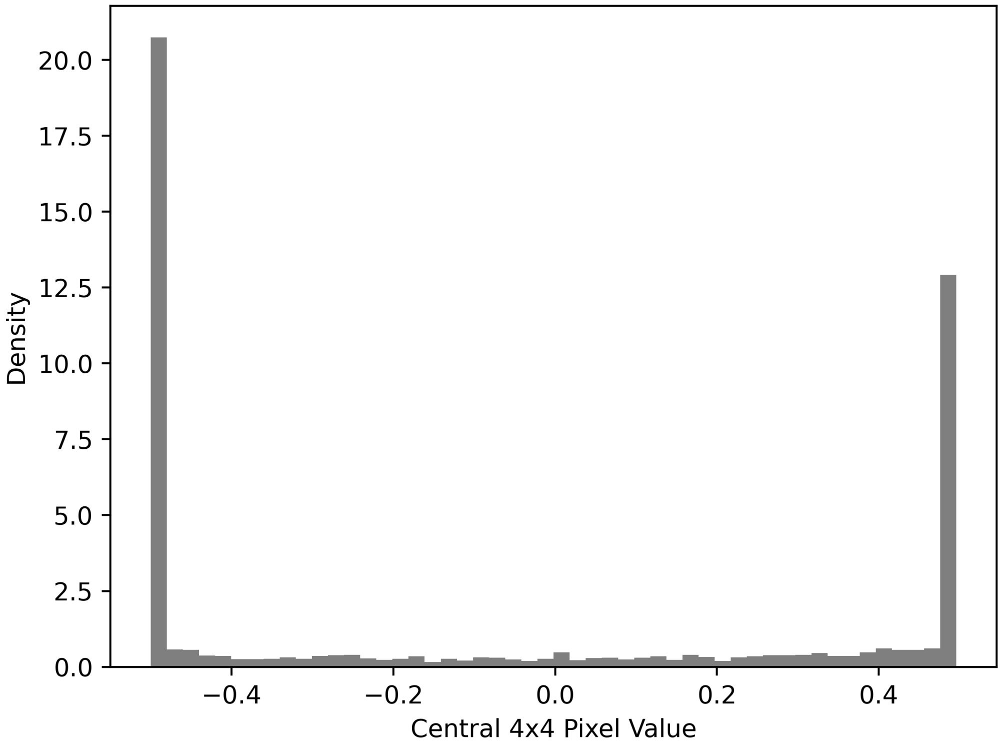

Figure 8: Density histogram of the central 4 × 4 pixel values of the MNIST dataset. The pixel values range from -0.5 to 0.5. The majority of the pixel values are concentrated around -0.5 (which corresponds to black pixels) and 0.5 (which corresponds to white pixels). The distribution is used to sample the initial seeds for the sampling process.

3.3 Results & Discussion

We train the Cold Zoom Diffusion model for 100 epochs with a learning rate of 2^{-4} and a batch size of 128. The decoder network $\mathbf{g}_t$ employed is identical to that used in the DDPM models, comprising 4 CNN blocks with channel sizes [16,32,32,16]. The small decoder network was chosen to limit the training time, and the epochs are the same with the DDPM model to allow for direct comparison. Fig.9 depicts the average loss per epoch for both training and testing sets. The downward trend in both train and test losses suggests that additional epochs could have further improved accuracy without causing overtraining. The train losses are higher than the train losses for the first few epochs because we are evaluating average loss per epoch. In constrast the test loss is calculated at the end of the epoch, when the model has been trained on the entire dataset.

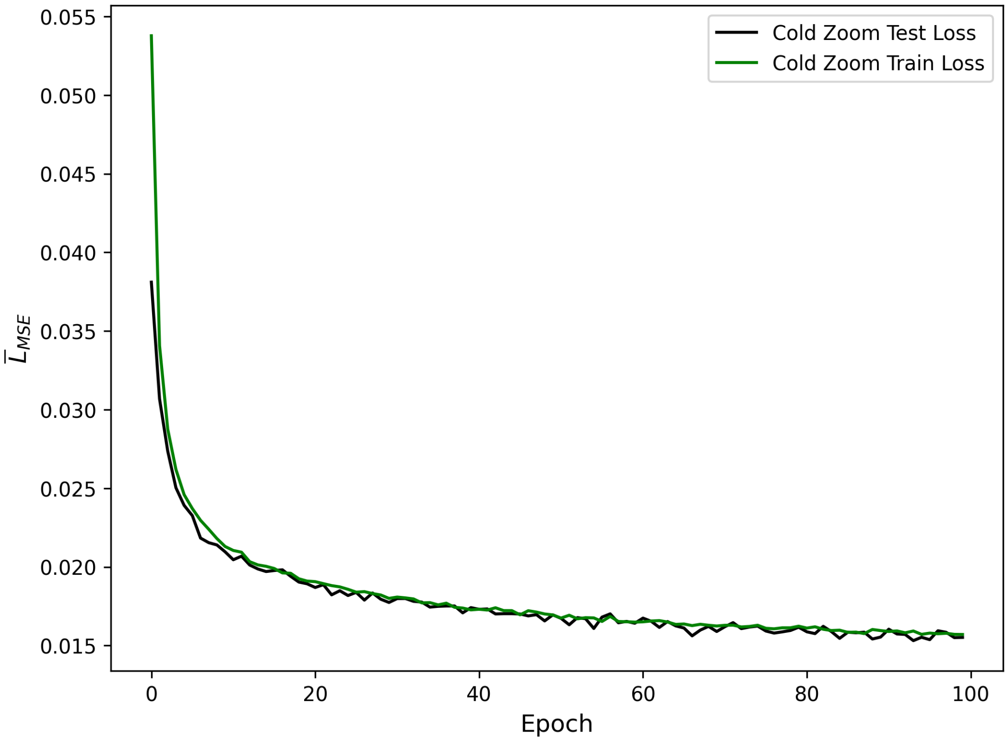

Figure 9: Mean squared error loss curves for the Cold Zoom Diffusion model. Depicted is the average loss per epoch for the training and test sets. While the test loss begins lower than the train loss, this is because the train loss is accumulated over the entire epoch, whereas the test is loss is cauculated at the end of the epoch.

We demonstrate the reconstructor model $\mathbf{g}_t$ by showcasing predictions for various zoom levels in Fig.10. The model, as expected, performs well at reconstructing images with relatively low degradation levels, but it stuggles to reproduce the original image starting from a fully degraded zoom of 24, as shown in Fig.11. For instance, the digit 5 is often misinterpreted as a 0. The model exhibits some bias towards reconstructing certain numbers, particularly 0s and 9s when most of the central pixels are black.

For the purposes of sampling however, the model does not start from a fully degraded image, but rather from a resized image of 16 independently sampled pixels. This approach reduces bias and introduces greater variability in predictions, but it also results in samples that do not always represent digits. We illustrate progressive de-zooming of generated seeds $\tilde{\mathbf{z}}_T$ that leads to samples $\hat{\mathbf{x}}$ during the sampling process in Fig.12. These images represent the latent variables $\tilde{\mathbf{z}}$ at every third step in the reverse process (FOR loop) of Algorithm 2.

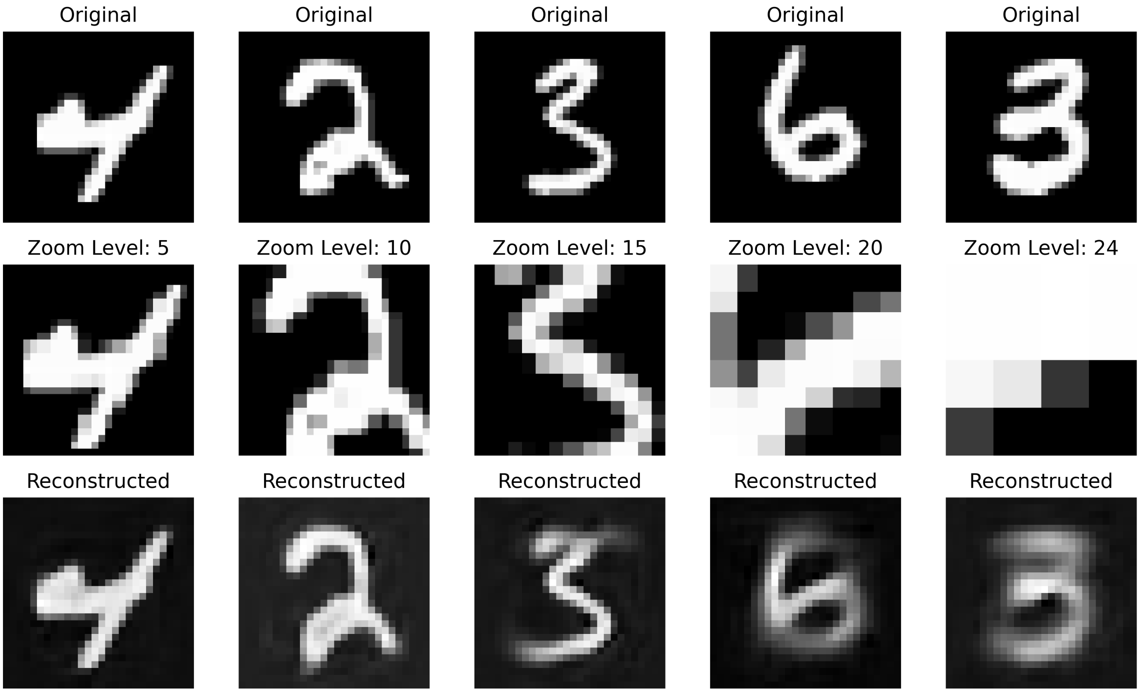

Figure 10: Reconstructor network predictions for various levels of zoom degradation. Note that zoom level 24 is a fully degraded image. In the sampling process, these predictions would then be further degraded and recondstructed sequentialyl.

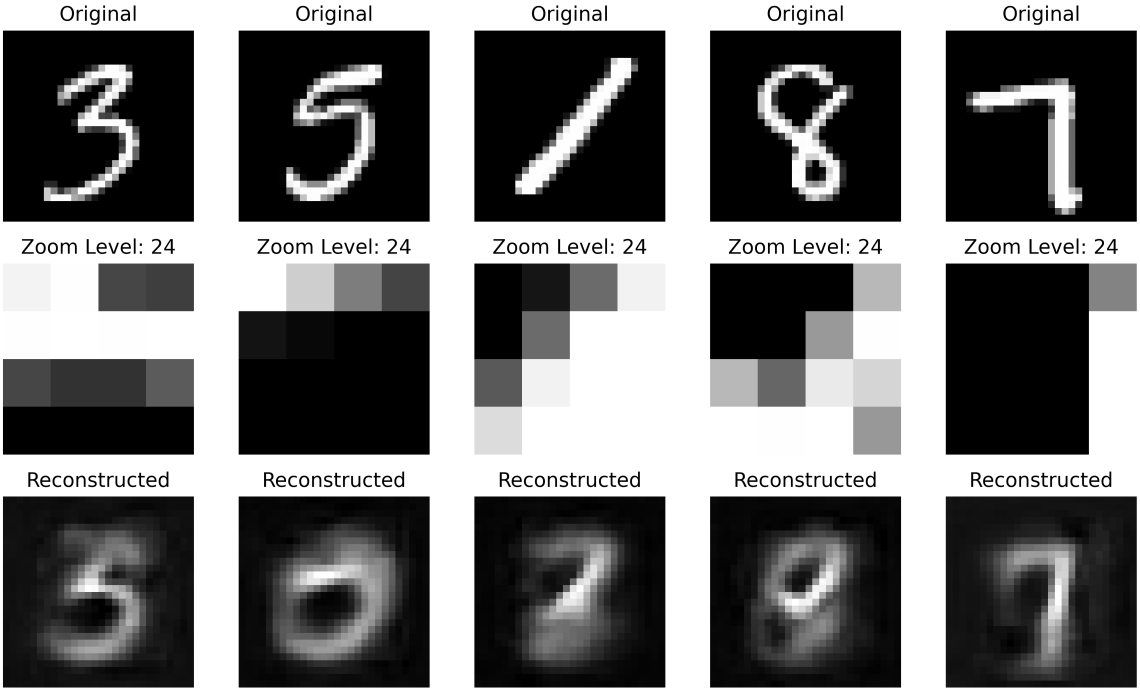

Figure 11: Reconstructor network predictions for fully degraded MNIST images at zoom level 24. The reconstructor makes analogous predictions from seeds in the first step of the sampling process. The digit 5 is mistakenly reconstructed as a 0.

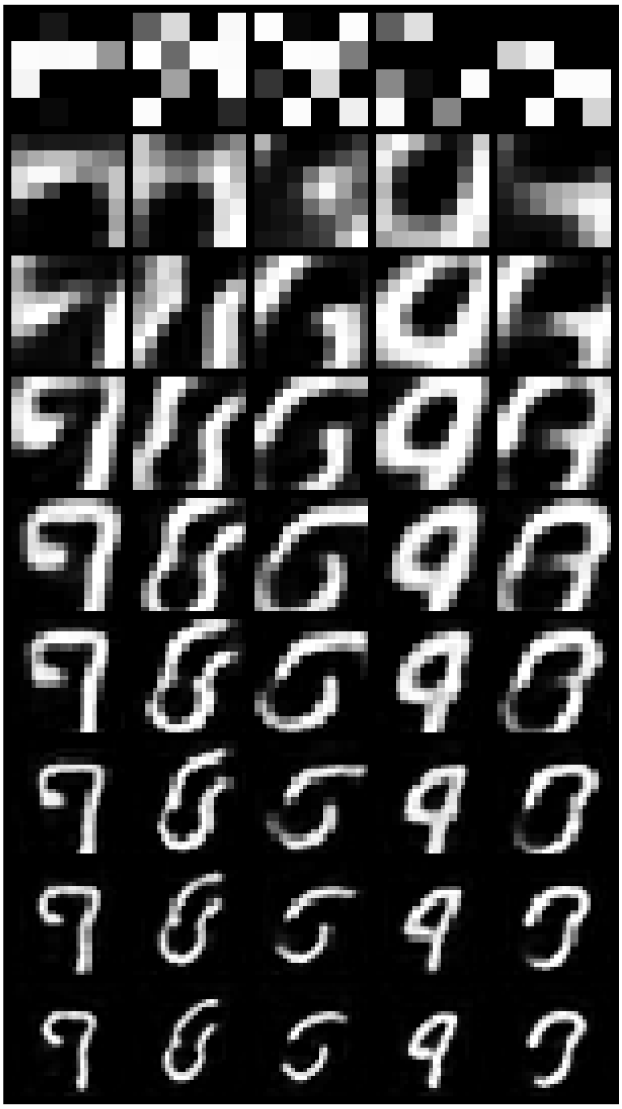

Figure 12: Evolution of the latent variables $\tilde{\mathbf{z}}$ during the sampling process, starting from random seeds $\tilde{z}_T$ at the top and ending with the generated samples $\hat{\mathbf{x}}$ at the bottom. The latent variables are shown every 3 steps in the reverse process. Between each step, a reconstruction prediction is made using $\mathbf{g}_t$. which is then degraded to the next step.

The FID scores of the Cold Zoom Diffusion model are compared with those of the DDPM models in Fig.13. Interestingly, the ZOOM diffusion model achieves comparable FID scores to the linear DDPM model. Although the FID scores of Zoom Cold Diffusion are on par with those of the DDPM models, they do not capture the variability of the image generation. This limitation in Zoom Cold Diffusion stems from the absence of noise addition in each step (the reverse process if fully deterministic), as opposed to the DDPM model. Nevertheless, it performes surprisingly well given the small scale of the CNN in the reconstructor network. Future work could involve using a custom classifier specifically trained on MNIST images to evaluate the quality of the generated samples. This classifier's embeddings could then by used for calculating the (FID) score in place of `inspectionv3`, potentially providing a more accurate reflection of sample quality of the MNIST dataset

Some high quality and low quality samples generated by the Cold Zoom Diffusion model are shown in Fig.14. Here we can see the limitations of the deterministic nature of the model. Although the initial seed is randomly sampled during the generation process, the resulting samples show a bias towards digits with predominantly dark or completely white central pixels. This bias reflects the distribution of central pixels in the MNIST dataset. Unlike the DDPM models, we do not add noise to the samples during the sampling process, which limits the variability of the samples generated. For this reason, the model is biased towards certain numbers such as 1, 8, and 9. Additionally, some initial seeds $\tilde{z}_T$ are not representative of typical MNIST images, leading to outputs that do not resemble digits. This is because by sampling independently for each pixel, we do not take into account the spatial correlation between pixels in the MNIST dataset.

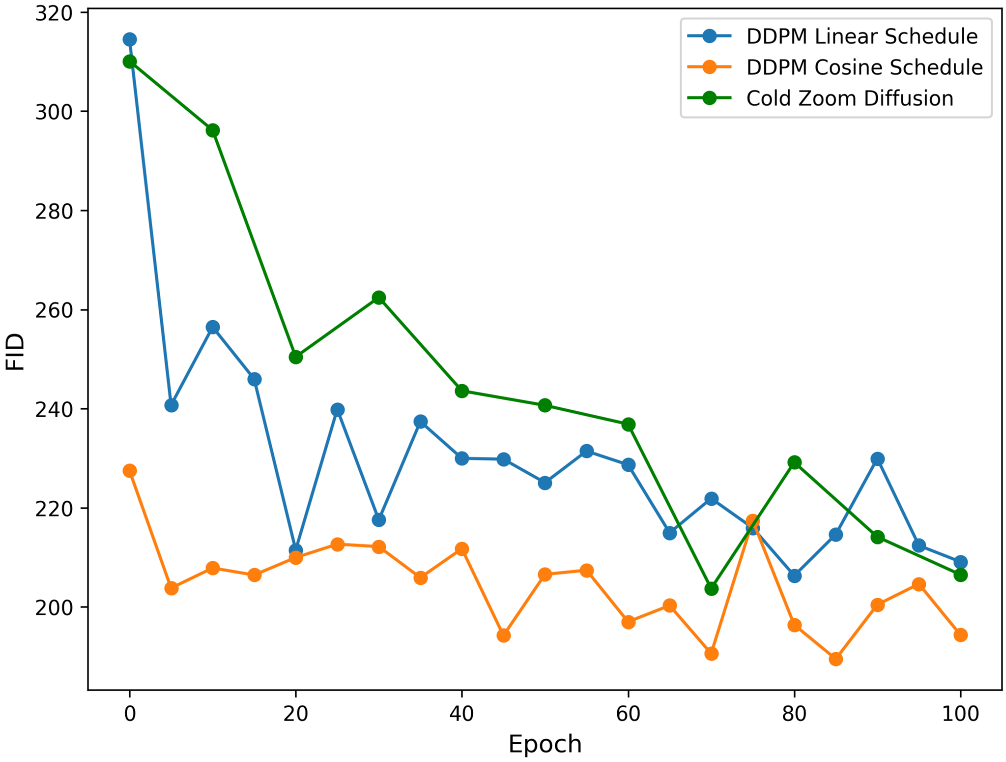

Figure 13: FID scores for the DDPM models and the Cold Zoom Diffusion model. The Cold Zoom Diffusion model achieves comparable FID scores to the linear DDPM model in the final epochs. The cosine DDPM model performs the best, with the lowest FID score.

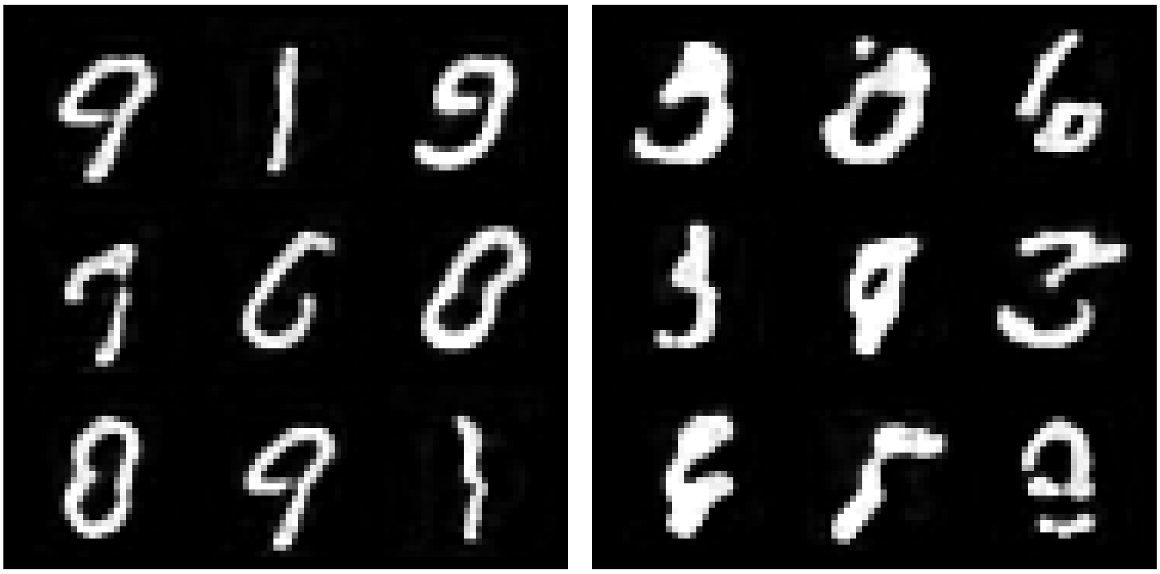

Figure 14: High quality samples (left grid) and low quality samples (right grid) generated by the Cold Zoom Diffusion model. The high quality samples show a bias towards particular digits, such as 1, 8, and 9. Noteably absent are the digits 2, 3, 4, 5. The low quality samples do not resemble digits, but retain the form of symbols.

4 Conclusion

In this study, we have implemented a Denoising Diffusion Probabilistic Model (DDPM) with both linear and cosine noise schedules. Additionally, we introduced a novel degradation strategy, termed Cold Zoom Diffusion. We trained these models on the MNIST dataset and evaluated their performance using the Fréchet Inception Distance metric. The cosine schedule DDPM model outperformed the linear schedule model in terms of both loss and FID score. The Cold Zoom Diffusion model achieved comparable FID scores to the linear DDPM model, despite its deterministic nature and small-scale reconstructor network. However, it showed a bias towards certain digits, a consequence of its deterministic degradation process. Future work could involve using a more sophisticated reconstructor network, such as a UNET architecture, to improve the quality of the generated samples. Improvements in the sampling methodology, such as integrating noise into the generation process, could also be explored to increase sample variability.

References

[1] Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie S. Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise, August 2022. arXiv:2208.09392 [cs].

[2] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[3] Alex Nichol and Prafulla Dhariwal. Improved Denoising Diffusion Probabilistic Models, February 2021. arXiv:2102.09672 [cs, stat].

[4] Naresh Babu Bynagari. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *Asian Journal of Applied Science and Engineering*, 8(1):25–34, April 2019.

[5] Frechet Inception Distance (FID) — PyTorch-Metrics 1.3.2 documentation.

[6] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision, December 2015. arXiv:1512.00567 [cs].

[7] Sadeep Jayasumana, Srikumar Ramalingam, Andreas Veit, Ayan Chakrabarti, Sanjiv Kumar, and Daniel Glasner. Rethinking FID: Towards a Better Evaluation Metric for Image Generation.

[8] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016.

[9] Dianyuan Han. Comparison of Commonly Used Image Interpolation Methods. *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering*, March 2013.