Prepared for submission to University of Cambridge

M2: Applications of Machine Learning - Coursework Assignment

Adnan Siddiquei

University of Cambridge

E-mail: as3438@cam.ac.uk

Contents

1 Overview of Diffusion Model Implementation 1

1.1 coursework_starter.ipynb 1

1.1.1 The Data 1

1.1.2 The CNN Model 1

1.1.3 The DDPM Model 2

1.1.4 The training loop 3

2 Training the model 3

3 Custom Degradation - Gaussian Blur 7

3.1 Model Design 7

A MNIST Classifier Details 10

1 Overview of Diffusion Model Implementation

1.1 coursework_starter.ipynb

The code provided in the coursework_starter.ipynb notebook trains a regular denoising diffusion probabilistic model (DDPM) on the MNIST dataset, using the `pytorch` library. This section provides a detailed explanation of how the given coursework_starter.ipynb code works. The notation used in this section adheres to the conventions used in Ch.18 of Understanding Deep Learning, Prince [1].

1.1.1 The Data

The code loads the MNIST dataset (60000 images) and preprocesses it with `transforms.toTensor` which converts the image to a tensor and scales it to the range $[0, 1]$, followed by `transforms.Normalize((0.5,),(1.0))` which shifts the range to $[-0.5, 0.5]$. The scaling and range shift can help improve the training dynamics. Given that MNIST images are grayscale (a single channel), the input dimensions are 1x28x28. The dataset is then loaded into a `DataLoader` object with a batch size of 128, yielding 468 batches per epoch.

1.1.2 The CNN Model

The code defines a Convolutional Neural Network `CNN` class which is the neural network that is used to learn the diffusion process - the noise that is added to the image at each step. This is instantiated with 5 hidden layers, each outputting: 16, 32, 32, 16, 1 channels respectively, with a kernel size of 7 at the first 4 layers, and a kernel size of 3 at the last layer. Each of the first 4 hidden layers (defined in the `CNNBlock` class) are composed of a 2D convolutional layer (with 0-padding on the edges) followed by a normalisation layer (`LayerNorm` which normalises each input to standard normal) and a GELU activation function. The normalisation stabilises the activations and places them in the transition range of the GELU function, which allows for full use of the GELU's characteristics.

Decoder models for the diffusion process can either be a set of multiple models, each trained to revert the noise at a particular step, or a single model trained to revert the noise at

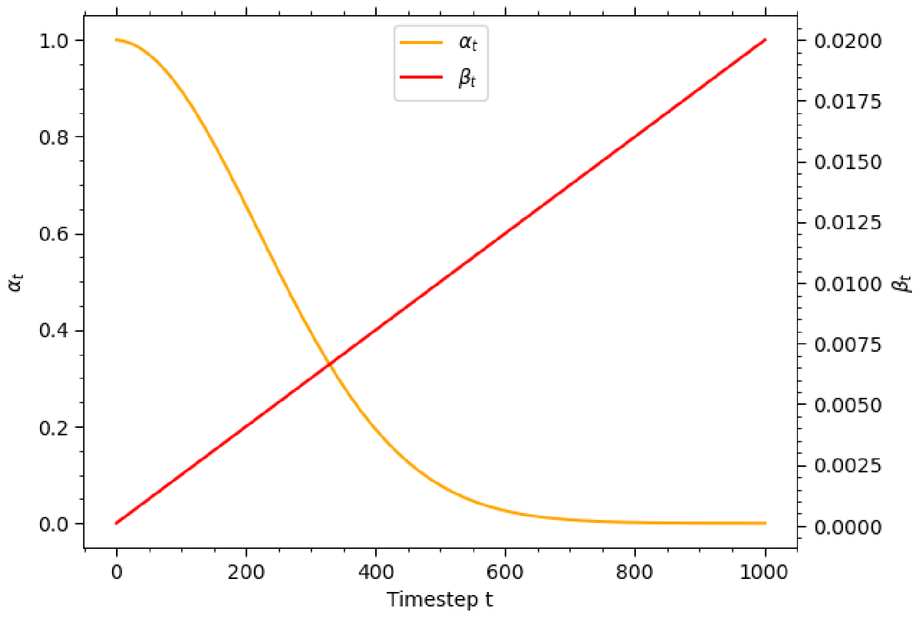


Figure 1: A plot of β_t and α_t for the linear noise schedule used in coursework_starter.ipynb. This figure illustrates that MNIST images transition for noiseless to standard normally distributed noise over 1000 steps, with most of the noise being added by about the 600'th step.

any given step by injecting information about the time step into the model. This CNN is the latter, and the `forward` function adds a time step dependent embedding to the pre-activations of the second layer so that the CNN can learn to revert the noise at any given step.

1.1.3 The DDPM Model

The training and sampling process is defined within the `DDPM` class, which contains a `forwar d` function that defines the forward pass of the model and a `sample` function that generates new MNIST data samples by iterating backwards through the diffusion process. The DDPM is instantiated with the CNN defined above and a linear noise schedule with β_t in the range $[10^{-4}, 0.02]$ across 1000 steps, as shown in Figure (1), where β_t and α_t are as defined in Ch.18 Prince [1].

The `forward` function samples 128 (the batch size) random α_t values from the noise schedule and a standard normally distributed noise tensor of shape `(128,1,28,28)`, and degrades every image in the batch with the noise according to Equation (1.1) [1].

$$z_t = \sqrt{\alpha_t} \cdot \mathbf{x} + \sqrt{1 - \alpha_t} \cdot \epsilon \tag{1.1}$$

Figure (2) illustrates the diffusion process. This batch of 128 noisy MNIST images (degraded to different extents) is then passed through the CNN (along with the corresponding time steps) to make a prediction on the noise that was added to the image at the corresponding step. The function returns the loss between the actual noise and this predicted noise as the mean squared error between the two.

The `sample` function can generate new MNIST samples by iterating backwards through the diffusion process using the learned parameters of the CNN, and a random sample of standard normal noise. Figure (3) illustrates the generation process.

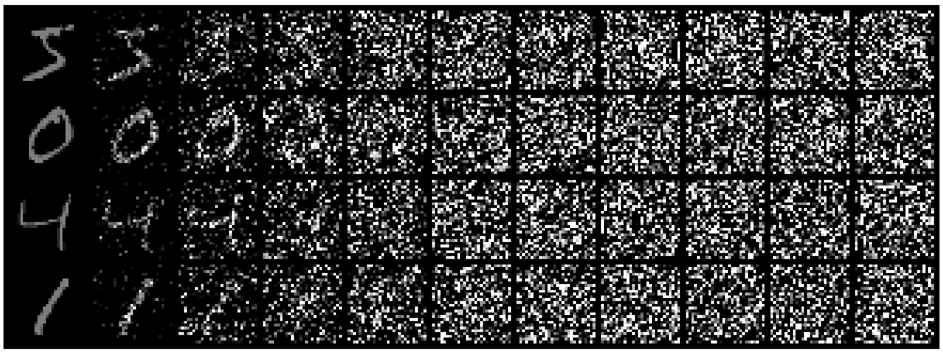


Figure 2: An illustration of the linear noise schedule applied to the first 4 MNIST images in the training datastet. The first column shows the original 28x28 image, and the subsequent 10 columns show the image after 100 steps of the diffusion process. The final column shows the image after 1000 steps.

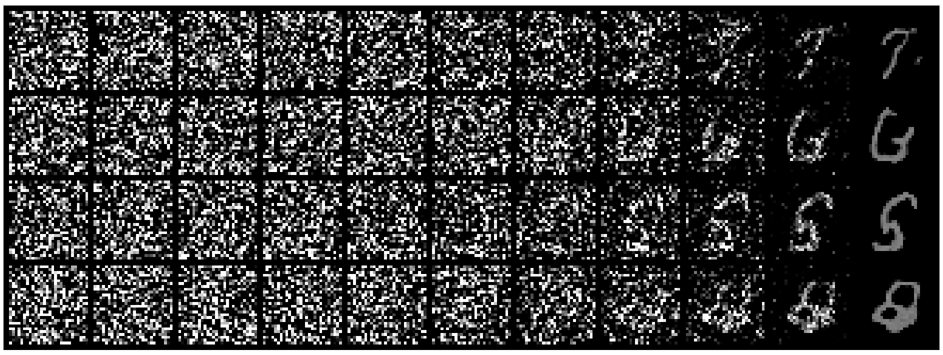


Figure 3: An illustration of the generation process. This is the generation of 4 new MNIST samples from the model trained with the default parameters in the coursework_starter.ipynb notebook. The first column shows the randomly sampled noise, the next 10 columns illustrate the image after 100 steps of decoding, with the final column showing the final generated image.

1.1.4 The training loop

The training loop iterates over 100 epochs, using the Adam optimiser and a learning rate of 2×10^{-4}, saving generated samples from the model at every epoch along with the loss at each epoch.

2 Training the model

This section discusses the training of the DDPM model with varying linear noise schedules.

Three different models were trained, the first model (default model) was trained with the provided noise schedule: $\beta_t = [10^{-4}, 0.02]$ with 1000 steps; the second model (the "long model") was trained with a longer noise schedule: $\beta_t = [10^{-4}, 0.004]$ with 5000 steps which added less noise at each step; and the third model (the "short model") was trained with a shorter noise schedule: $\beta_t = [10^{-4}, 0.1]$ with 200 steps which added more noise at each step. The number of steps was amended for each model such that the final noise level was the same for each model (α_t of the order of 10^{-5} at the final time step). All models were also trained for 100 epochs with a batch size of 128, with the Adams optimiser, a learning rate of

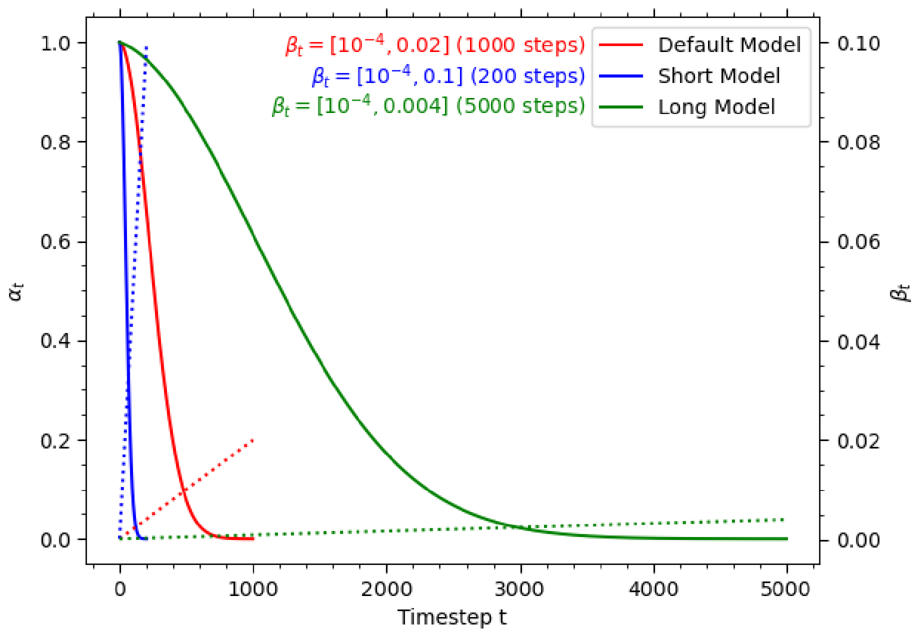


Figure 4: A plot of each noise schedule that was evaluated. The dotted lines represent the β_t values and the solid lines represent the α_t values.

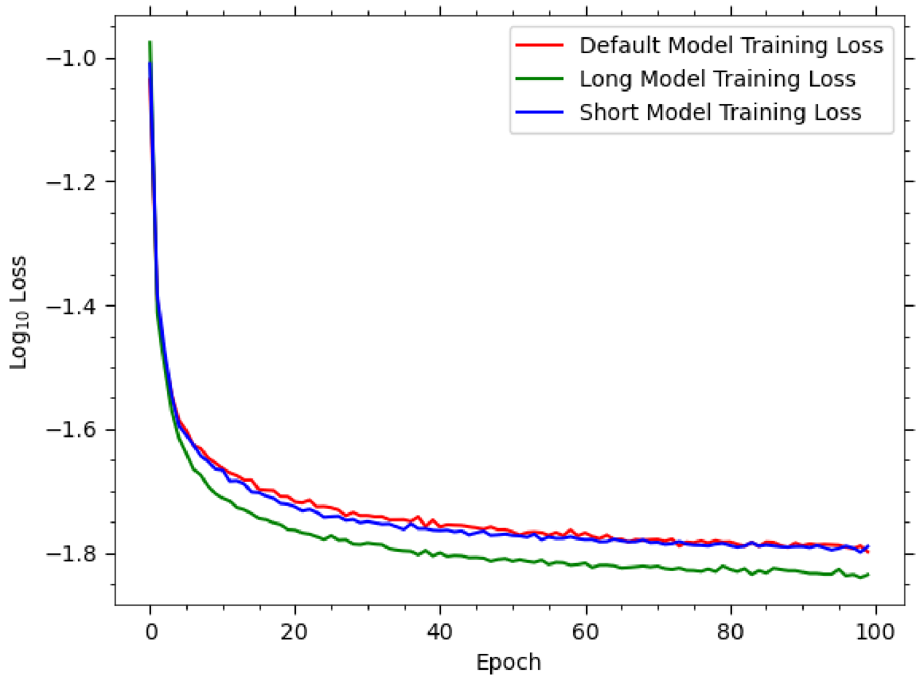
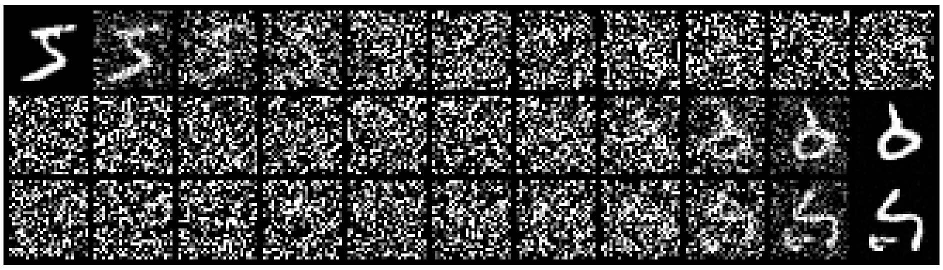
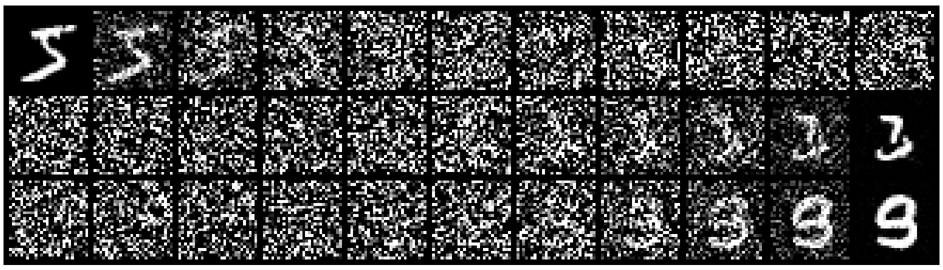


Figure 5: A plot of the training loss for each model over 100 epochs.

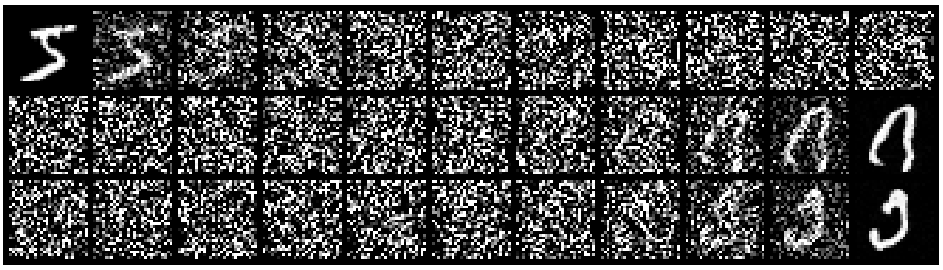
2×10^{-4}. The CNN architecture for all models was identical to as described in Section (1), with exception that the number of `CNNBlocks` was increased by 1 (from $[16, 32, 32, 16, 1]$ to $[16, 32, 64, 32, 16, 1]$ where the numbers indicate the number of channels in each layer). This



(a) The encoding and decoding process for the default model, with each column representing equal time steps apart (100 time steps).



(b) The encoding and decoding process for the short model, with each column representing equal time steps apart (20 time steps).

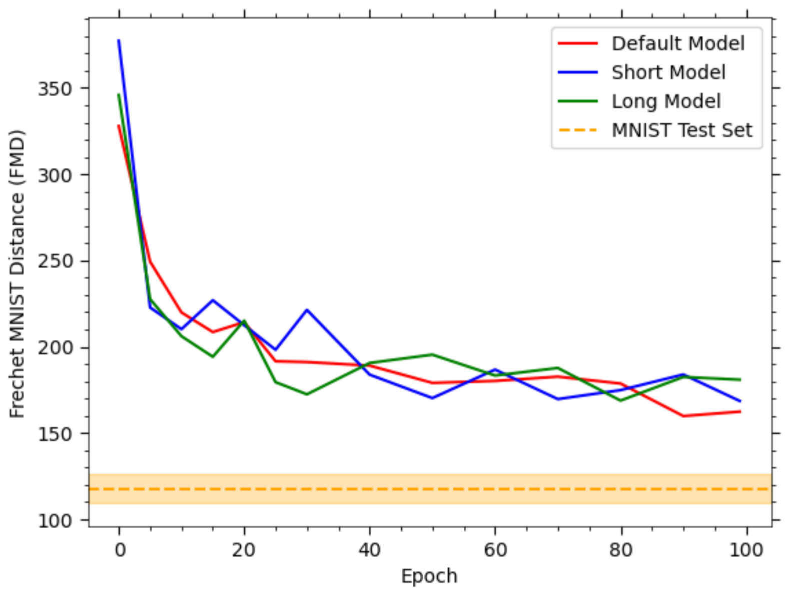


(c) The encoding and decoding process for the long model, with each column representing equal time steps apart (500 time steps).

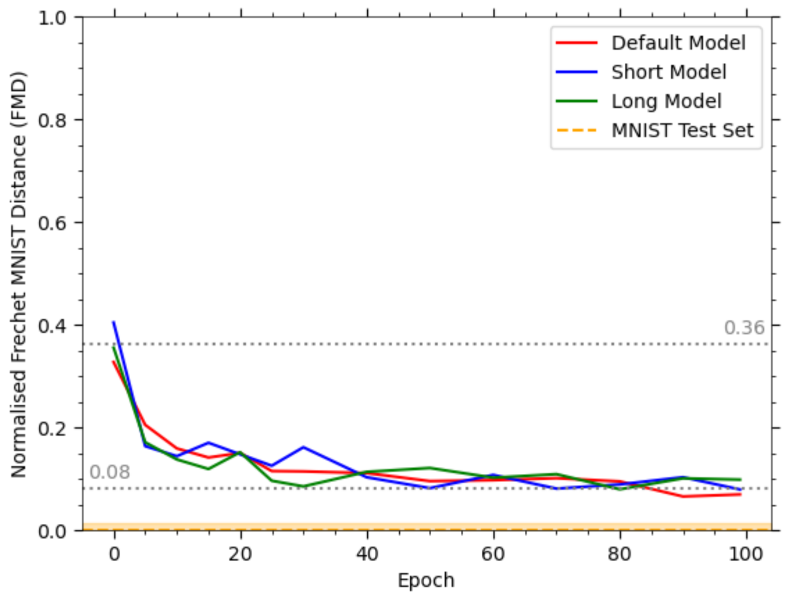
Figure 6: The encoding and decoding process for each of the models trained in Section (2).

was increased whilst considering the trade-off between fidelity of the generated samples and the computational cost of training. Additionally, the `transforms.Normalize((0.5,),(1.0))` step was removed from the MNIST dataset transformation to allow for more equivocal comparison between the these models and the custom degradation model implemented in Section (3) (more on this later).

Figure (4) illustrates the noise schedules used for each model, Figure (5) shows the training loss for each model over 100 epochs, and Figure (6) illustrates the encoding and decoding process for all 3 models. Figure (7) shows the Frechet MNIST Distance (FMD) for the generated samples from each model as the model is trained over the 100 epochs. The FMD is an adjusted version of the Frechet Inception Distance (FID) which uses an MNIST classifier (that was trained for this purpose) with a 1024-length feature vector at the penultimate layer to calculate the Frechet distance instead of the InceptionV3 model. See Appendix (A) for more details on the MNIST classifier that was used to calculate the FMD. The FMD was used instead of the FID as the MNIST dataset is a narrow context dataset and so a model trained specifically on this dataset would be more appropriate for calculating the Frechet distance.

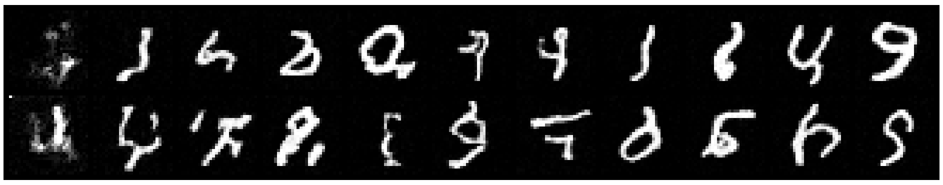


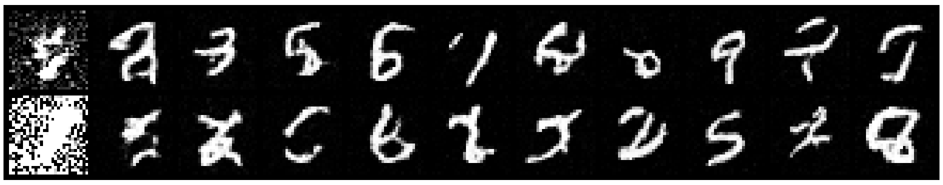
(a) The raw FMD values for each model, with the orange line showing mean and standard deviation of the FMD value for 15 random samples from the MNIST test set.



(b) The same FMD values as above, but normalised between 0 and 1, where 0 is the mean FMD value for the MNIST test set (as shown on the above figure) and 1 is the FMD value for a completely black image. The grey dotted lines show the mean start and end values.

Figure 7: Two plots of the Frechet MNIST Distance (FMD) for the generated samples from each model over 100 epochs. The top plot is normalised, the bottom is not. At each plotted epoch, 32 samples were generated from the model and compared against the first 512 samples from the MNIST test dataset.


(a) The default model.


(b) The short model.

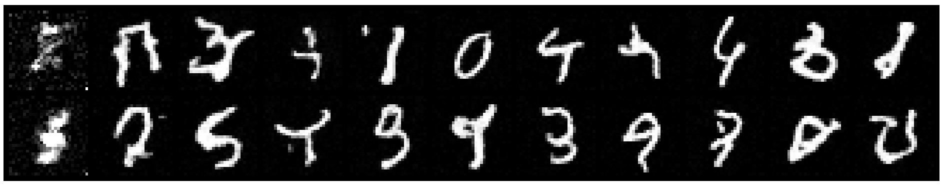

(c) The long model.

Figure 8: The samples generated by each model at every 10 epochs. Epoch 1 is on the left, and epoch 100 is on the right.

From these figures, several conclusions can be drawn. All models show convergence in their training loss and converge to similar values, with very little instability in the training. The FMD score falls in a correlated fashion to the training loss, indicating that the models are indeed generating samples that are statistically similar to the MNIST dataset. However, neither model outperforms the others by any significant means across either of these metrics. Figure (7b) indicates a 4.5x mean decrease in the FMD values, however on inspection to Figure (8) and Figure (9), the samples generated by these models are still fairly poor, and so a normalised FMD value of 0.08 is still not particularly impressive.

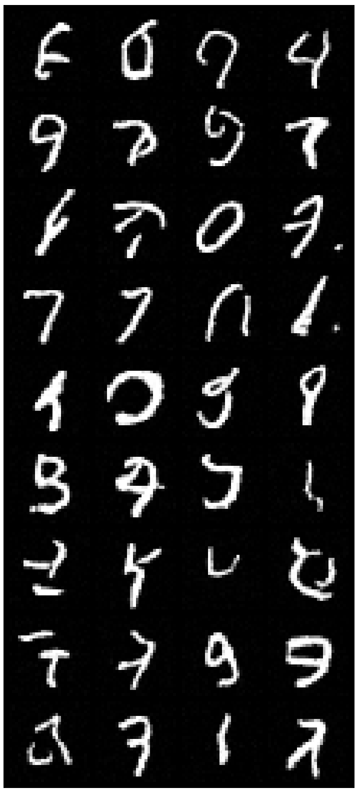
3 Custom Degradation - Gaussian Blur

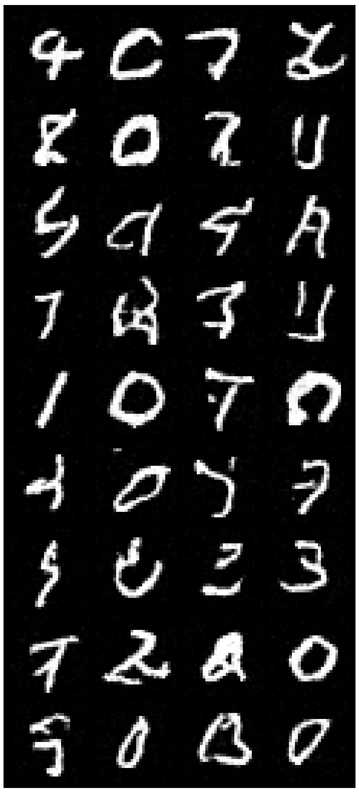
A Gaussian blur degradation was implemented to train a diffusion model on the MNIST dataset. This section discusses the design of this model, the training process and a comparison of this model with the 3 models trained in Section (2).

3.1 Model Design

The exact same CNN architecture was used as in Section (2) to learn the diffusion process. The primary differences were the `forward` and `sample` functions, with the sampling process derived from Bansal et al., (2022) [2].

The forward function worked in a similar way to the `DDPM` models. A random timestep t was sampled in the range $[1, 20]$ and a Gaussian blur was applied to the MNIST image using `torchvision.transforms.GaussianBlur` with a kernel size of 29 and a standard deviation of according to the linear "blur schedule" shown in Figure (10). As displayed in the figure, the standard deviation of the Gaussian blur increased linearly with time, resulting


(a) The default model.


(b) The short model.

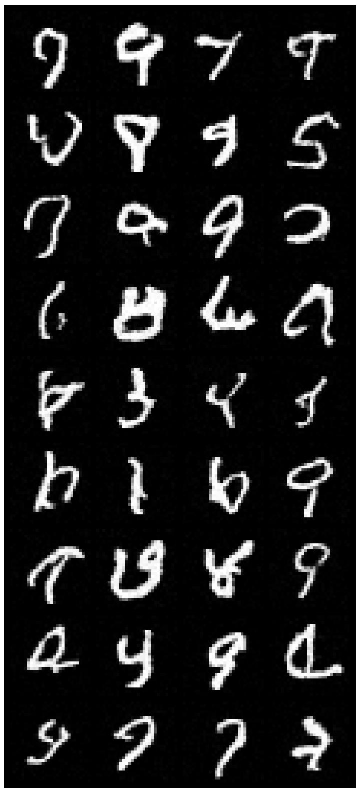

(c) The long model.

Figure 9: 36 samples generated by each model at the final epoch.

in a exponentially blurred image as the timestep increased. Given that the `GaussianBlur` function 0-pads on the edge, the images were scaled to $[0, 1]$ to ensure that 0 represented black cells. The loss function was an `nn.MSELoss`, with arguments being the original image and the blurred image, and the model was trained using the MNIST training set. All other hyperparameters (epochs, optimiser, learning rate) were kept the same as in Section (2).

Conditional sampling was used for the sampling process where a random MNIST digit from the test set was blurred to the final timestep, and the model was used to reverse the blurred image back to a new image by using Algorithm 2 from Bansal et al., (2022) [2].

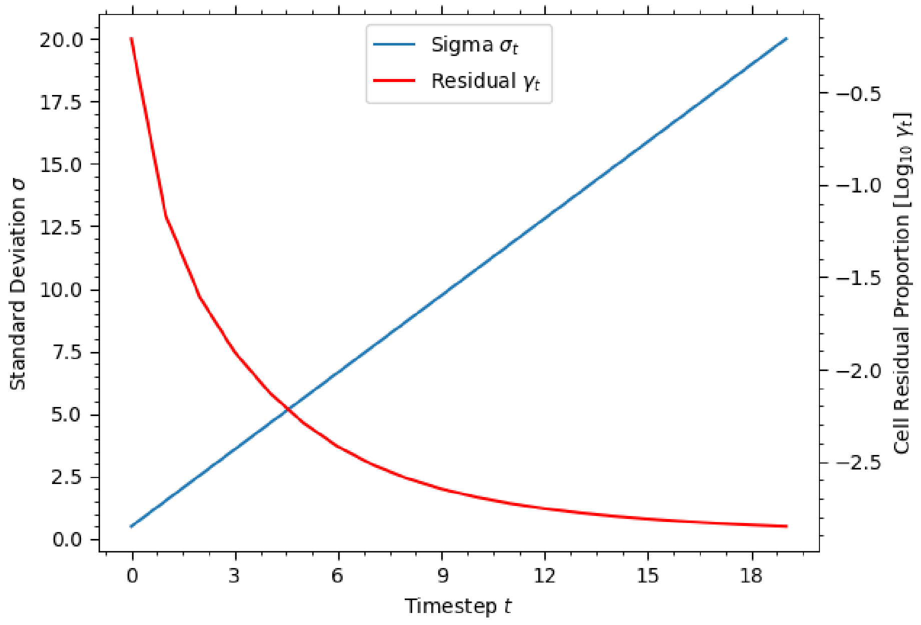


Figure 10: A plot of the blur schedule for the Gaussian blur mode. The blue line shows how the standard deviation of the Gaussian blur changes across timesteps (left axis). The red line shows the proportion of the original cell's value that is retained in the blurred cell (computed from the 1 cell central area of the 2D Gaussian kernel with a kernel size of 29). The right axis is in log scale and is termed γ_t.

A MNIST Classifier Details

The MNIST classifier model is a CNN composed of 4 `CNNBlocks` ($[32, 64, 128, 64]$ channels per block) following by aa `nn.AdaptiveAvgPool2d` to lower the dimensionality of the penultimate feature vector to 1024 before it is fed into a fully connected `nn.Linear` layer with 10 output units (one for each class in the MNIST dataset). This classifier was trained for XXX epochs with a batch size of 128 and a learning rate of 2×10^{-4} using the Adam optimiser. The loss function used was the cross-entropy loss function and the test set classification accuracy was reached XXXX on the final epoch. To extract the 1024-length feature vector, the final `nn.Linear` layer was set to `nn.Identity` so that the model outputted the penultimate layer feature vector instead of the class predictions. The model was trained on the MNIST training set and the test set was used for evaluation of the FMD.

References

[1] Simon J.D. Prince, *Understanding Deep Learning*. The MIT Press, 2023. Available at https://udlbook.github.io/udlbook/ [Accessed: 20-Mar-2024].

[2] Bansal et al., *Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise*. University of Maryland and New York University, 2022. Available at https://arxiv.org/pdf/2208.09392.pdf [Accessed: 28-Mar-2024].