UNIVERSITY OF
CAMBRIDGE

DiS
MPHIL

Applications of Machine Learning

Coursework Submission

A REPORT PRESENTED
BY
STEVEN DILLMANN

Departments
Department of Applied Mathematics and Theoretical Physics
Department of Physics (Cavendish Laboratory)
Institute of Astronomy

Degree
MPhil Data Intensive Science

Module
M2 Applications of Machine Learning

Supervision
Dr Miles Cranmer

ST JOHN'S COLLEGE
UNIVERSITY OF CAMBRIDGE
17TH DECEMBER 2023

List of Figures

1 Gaussian degradation for $\alpha_t = 1.000, 0.970, 0.896, 0.522, 0.078, 0.003$ and $4.077 \cdot 10^{-5}$. 3
2 Evolution of training and validation loss with the number of epochs for Model 1 and 2. 4
3 FID scores for Model 1 and 2 at three different points of the training process. 5
4 Evolution of generated digit images during the training process for Model 1. 6
5 Evolution of generated digit images during the training process for Model 2. 6
6 Distribution of predicted digit classes for 1000 generated images from Model 1 and 2. 7

7 Impulse noise density ρ_t throughout the cosine schedule. 8
8 Impulse degradation for $\rho_t = 0.050, 0.063, 0.100, 0.150, 0.200, 0.237$ and 0.250. 9
9 Images at different epochs from training Model 3 displaying very low diversity. . . . 10
10 Evolution of training and validation loss with the number of epochs for Model 4. . . 11
11 FID scores for Model 4 at three different points of the training process. 12
12 Distribution of predicted digit classes for 1000 generated images for Model 3. 12
13 Evolution of generated digit images during the training process for Model 4. 12
14 FID scores for Model 1, 2 and 4 at three different points of the training process. . . . 13
15 Visual quality of generated digits 0 to 9 for Model 1, 2 and 4 as predicted by the MNIST classifier. Note that the coloured structures in the images are due to an error in saving the images appropriately. 14

List of Tables

1 Comparison of hyperparameters for Model 1 and 2. 4
2 Comparison of IS for Model 1 and 2. 5

3 Comparison of hyperparameters for Model 3 and 4. 10
4 IS results for Model 4. 11
5 Comparison of IS for Model 1, 2 and 4. 13
6 Comparison of FID for Model 1, 2 and 4. 13

Contents

List of Figures **ii**

List of Tables **iii**

1 Training a Diffusion Model **1**

1.1 Part 1(a) 1

1.1.1 Diffusion Model 1

1.1.2 Noise Schedule & CNN Predictive Model 3

1.2 Part 1(b) and 1(c) 4

1.2.1 Model Definition 4

1.2.2 Training and Validation 4

1.2.3 Inception Score 5

1.2.4 Fréchet Inception Distance 5

1.2.5 Visual Quality and Diversity 6

1.2.6 Analysis and Discussion 7

2 Custom Degradation **8**

2.1 Part 2(a) 8

2.1.1 Degradation Strategy 8

2.1.2 Training and Sampling Algorithms 9

2.2 Part 2(b) 10

2.2.1 Model Definition 10

2.2.2 Training and Validation 11

2.2.3 Image Quality and Diversity Metrics 11

2.2.4 Analysis and Discussion 11

2.3 Part 2(c) 13

2.3.1 Gaussian Noise vs Impulse Noise Degradation 13

2.3.2 Summary and Conclusions 15

A Use of auto-generation tools **18**

A.1 Co-Pilot 18

A.2 ChatGPT 18

A.2.1 Prompt 1 18

Total LaTeX Word Count: 2998/3000 (excluding Figures, Tables, Bibliography and Appendix)

1 Training a Diffusion Model

1.1 Part 1(a)

1.1.1 Diffusion Model

The notebook `coursework_starter.ipynb` implements a denoising diffusion probabilistic model (`DDPM`) [1], and training loop on the MNIST dataset [2] in Pytorch [3]. Diffusion models [4] are generative latent variable models [5] that consist of an encoder mapping data through a series of noise transformations, and decoder that aims to reconstruct the original data by learning these probabilistic mappings. The theory and implementation presented below is mainly based on [6].

Forward Diffusion Process (Encoder)

The encoder takes a sample from a real data distribution $\mathbf{x_0} \sim p(\mathbf{x}_0)$, mapping it through a chain of latent variables $\mathbf{x}_1, \ldots, \mathbf{x}_T$ with the same dimensionality as $\mathbf{x}_0$ with:

$$\mathbf{x}_t = \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1} + \sqrt{\beta_t} \cdot \boldsymbol{\epsilon}_t \qquad \forall t \in \{1, \ldots, T\}, \tag{1}$$

where $\boldsymbol{\epsilon}_t$ is Gaussian noise gradually added according to the noise schedule $\{\beta_t \in [0,1]\}_{t=1}^T$. We can write the forward process in terms of a Markov chain [7] with the transition probabilities:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}_{\mathbf{x}_t}\left(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right). \tag{2}$$

The joint distribution of all latent variables $\mathbf{x}_1, \ldots, \mathbf{x}_T$ given the input sample $\mathbf{x}$ is:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}). \tag{3}$$

When $T \to \infty$, the probability $q(\mathbf{x}_T|\mathbf{x}_0) = q(\mathbf{x}_T)$ approaches a standard Gaussian. The decoder is trained to invert the above process using multiple samples $\mathbf{x}_t$ drawn from $q(\mathbf{x}_t|\mathbf{x}_0)$ at any time t using an analytical expression known as the diffusion kernel [6], where $\alpha_t = \prod_{i=1}^{t} 1 - \beta_i$. For $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}_{\mathbf{x}_t}\left(\sqrt{\alpha_t}\mathbf{x}_0, (1-\alpha_t)\mathbf{I}\right), \tag{4}$$

$$\mathbf{x}_t = \sqrt{\alpha_t} \cdot \mathbf{x}_0 + \sqrt{1-\alpha_t} \cdot \boldsymbol{\epsilon}. \tag{5}$$

Reverse Diffusion Process (Decoder)

If we reversed the forward process by sampling from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, we could reconstruct the data sample $\mathbf{x}$ from a Gaussian noise input $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$p(\mathbf{x}_T) = \mathcal{N}_{\mathbf{x}_t}(\mathbf{0}, \mathbf{I}). \tag{6}$$

The true reverse distributions $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are too complex too sample from as they need the entire dataset $Pr(\mathbf{x}_0)$. If β_t is small and T is large, $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is also a normal distribution. Therefore, we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ as Gaussians and use a model $\mathbf{f}_t[\mathbf{x}_t, \phi_t]$ that predicts their means:

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t, \phi_t) = \mathcal{N}_{\mathbf{x}_{t-1}}\left(\mathbf{f}_t[\mathbf{x}_t, \phi_t], \sigma_t^2 \mathbf{I}\right), \tag{7}$$

where the variances $\{\sigma_t^2\}$ are prespecified. The joint distribution of $\mathbf{x_0}$ and latents $\mathbf{x}_1, \ldots, \mathbf{x}_T$ is:

$$p(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p(\mathbf{x}_{t-1}|\mathbf{x}_t, \phi_t). \tag{8}$$

Loss Function and Training

Using Maximum Likelihood Estimation, we first find the likelihood function by marginalisation and finally maximise the log-likelihood of the training set $\{\mathbf{x}_{0_i}\}_{i=1}^{N}$ with respect to the parameters:

$$p(\mathbf{x}_0|\phi_{1:T}) = \int p(\mathbf{x}_{0:T})\, d\mathbf{x}_{1:T} \tag{9}$$

$$\hat{\phi}_{1:T} = \underset{\phi_{1:T}}{\text{argmax}}\left[\sum_{i=1}^{I}\log\, p\left(\mathbf{x}_0^i|\phi_{1:T}\right)\right]. \tag{10}$$

Maximising this directly is not possible. Further manipulation involving the evidence lower bound and reparameterisation of the target and network [1, 6] gives the loss function:

$$L[\phi_{1:T}] = \sum_{i=1}^{I}\sum_{t=1}^{T}\|g_t\left[\mathbf{x}_{t_i},\phi_t\right]-\epsilon_{t_i}\|^2 = \sum_{i=1}^{I}\sum_{t=1}^{T}\left\|g_t\left[\sqrt{\alpha_t}\cdot\mathbf{x}_{0_i}+\sqrt{1-\alpha_t}\cdot\epsilon_{t_i},\phi_t\right]-\epsilon_{t_i}\right\|^2, \tag{11}$$

where the new model $\hat{\epsilon} = g_t[\mathbf{x}_t,\phi_t]$ predicts the noise ϵ that was blended with the data $\mathbf{x}_0$ to create the current variable $\mathbf{x}_t$. This replaced the original model $\hat{\mathbf{x}}_{t-1} = \mathbf{f}_t[\mathbf{x}_t,\phi_t]$ according to:

$$\mathbf{f}_t[\mathbf{x}_t,\phi_t] = \frac{1}{\sqrt{1-\beta_t}}\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}g_t[\mathbf{x}_t,\phi_t] \tag{12}$$

Equation 11 is the cumulative MSE loss between the predicted and actual noise, computed with the function `DDPM.forward`. This loss can now be used to train the model with Algorithm 1.

Algorithm 1 Diffusion Model Training

```
procedure TrainingDDPM($p(\mathbf{x}_0)$)
    repeat
        $\mathbf{x}_{0_i} \sim p(\mathbf{x}_0)$                         ▷ Sample from dataset distribution $p(\mathbf{x}_0)$
        $t \sim \text{Uniform}(\{1,\ldots,T\})$                     ▷ Sample random time step $t$
        $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})$                                  ▷ Sample noise
        $\ell_i = \left\|g_t\left[\sqrt{\alpha_t}\mathbf{x}_{0_i}+\sqrt{1-\alpha_t}\epsilon,\phi_t\right]-\epsilon\right\|^2$        ▷ Compute individual loss
        Accumulate losses and perform gradient descent to update the model parameters $\phi_t$
    until converged
    return $\phi_t$
```

Sampling

Equation 12 can be used to obtain the prediction $\hat{\mathbf{x}}_{t-1} = \mathbf{f}_t[\mathbf{x}_t,\phi_t]$. To sample $\mathbf{x}_{t-1} \sim p(\mathbf{x}_{t-1}|\mathbf{x}_t,\phi_t)$ then means computing:

$$\mathbf{x}_{t-1} = \hat{\mathbf{x}}_{t-1} + \sigma_t\epsilon, \tag{13}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})$. Equation 13 can then be used for the complete sampling procedure with Algorithm 2 implemented in the function `DDPM.sample`.

Algorithm 2 Diffusion Model Sampling

```
procedure SamplingDDPM($g_t[\cdot,\phi_t]$, $\beta_t$, $\alpha_t$)
    $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0},\mathbf{I})$                                          ▷ Sample last latent
    for $t = T,\ldots,1$ do
        $\hat{\mathbf{x}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}g_t[\mathbf{x}_t,\phi_t]$    ▷ Predict previous latent with $g_t[\cdot,\phi_t]$, $\beta_t$, $\alpha_t$
        $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ if $t > 1$, else $\epsilon = 0$                  ▷ Draw new noise vector
        $\mathbf{x}_{t-1} = \hat{\mathbf{x}}_{t-1} + \sigma_t\epsilon$                          ▷ Add noise to previous latent
    end for
    return $\mathbf{x}_0$
```

1.1.2 Noise Schedule & CNN Predictive Model

Noise Schedule

A linear noise schedule is used in the function `ddpm_schedule`. The inputs are the start and end points, `beta1` and `beta2`, of the schedule. It outputs (i) the noise schedule $\{\beta_t \in [0, 1]\}_{t=1}^{T}$ defined as a linearly spaced sequence from `beta1` to `beta2` over T time steps, and (ii) $\alpha_t = \prod_{i=1}^{t} 1 - \beta_i$. The Gaussian noise degradation is visualised in Figure 1 for different α_t.

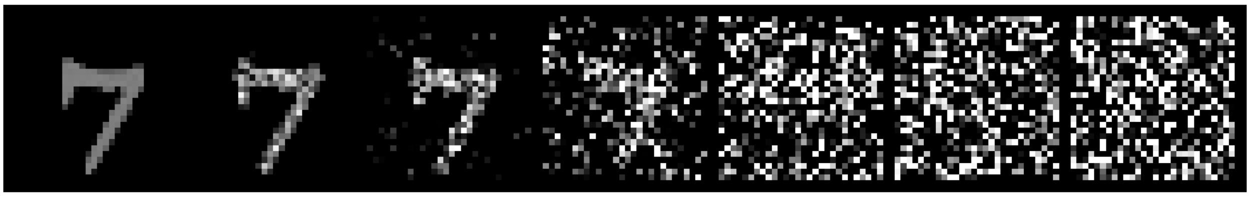


Figure 1: Gaussian degradation for $\alpha_t = 1.000, 0.970, 0.896, 0.522, 0.078, 0.003$ and $4.077 \cdot 10^{-5}$.

CNN Diffusion Process Model

The chosen noise prediction model $g_t[\mathbf{x}_t, \phi_t]$ is a 2D convolutional neural network (CNN) [8] defined by the class `CNN`. It consists of the following main components:

1. **Convolutional Blocks:** Individual CNN blocks, defined by the class `CNNBlock`, form the core of the CNN. Each block consists of a convolutional layer, normalisation layer for training stabilisation and a GELU [9] activation function to introduce smooth non-linearity. By default, there are three convolutional layers with kernel size 7 and padding 3. The first layer takes a single-channel (greyscale) input image of shape (28, 28) and produces 64 feature maps. The second layer produces 128 and the third 64 feature maps again. This allows the CNN to learn hierarchical representations effectively.
2. **Final Convolutional Layer:** A final convolutional layer adjusts the output scale and shape to the original input dimensions without applying an activation function to maintain the original representation of the learned features.
3. **Temporal Embedding Layer and Encoding:** The notion of time (t) in the context of diffusion models is introduced into the CNN with `time_embed` and `CNN.time_encoding`, allowing the model to process temporal information alongside spatial information during training.

1.2 Part 1(b) and 1(c)

In this part, we document the training process of two different models, evaluate their performance and analyse each trained model. The differences in the results are discussed in Section 1.2.6.

1.2.1 Model Definition

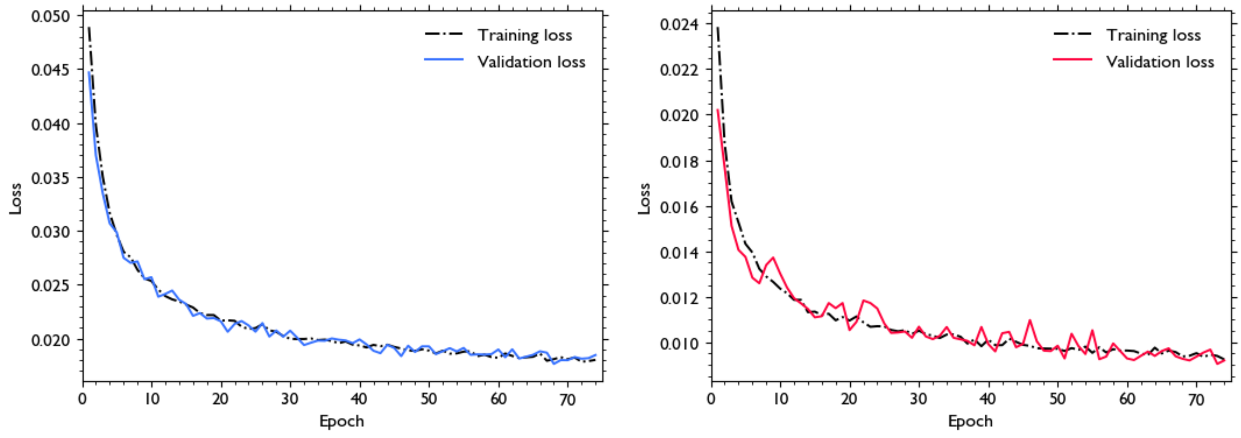
Both models have one input channel with image shape (28, 28), use GELU activations and Adam as an optimiser [10]. The differences between Model 1 and 2 are presented in Table 1.

Table 1: Comparison of hyperparameters for Model 1 and 2.

Hyperparameter	**Model 1**	**Model 2**
CNN Hyperparameters		
Hidden Units and Layers (`n_hidden`)	(16, 32, 32, 16)	(32, 64, 128, 64, 32)
Activation Function	GELU	GELU
DDPM Hyperparameters		
Noise Schedule (`beta1, beta2`)	(1e-4, 0.02)	(1e-4, 0.05)
Diffusion Steps T (`n_T`)	1000	1200
Training Hyperparameters		
Learning Rate	2e-4	1e-3
Optimizer	Adam	Adam

1.2.2 Training and Validation

Our MNIST dataset includes $N = 60,000$ images, set up with a batch size of 128 and 4 walkers. Using a 80:20 split gives a training set of size $N_{train} = 48,000$ and a validation set of size $N_{val} = 12,000$. Both models are trained for 75 epochs, while storing intermediate model weights. The training and validation loss curves are shown in Figure 2.



(a) Training process for Model 1.

(b) Training process for Model 2.

Figure 2: Evolution of training and validation loss with the number of epochs for Model 1 and 2.

For Model 1, the losses converge and plateau quickly, indicating that most learning occurs early on. The smallest validation loss is achieved at epoch 68 with $L_{val} = 0.018$. For Model 2, the loss curves also converge relatively quickly and to a lower loss, however the validation loss shows significant fluctuations. The smallest validation loss is achieved at epoch 73 with $L_{val} = 0.009$. The training times on a local machine are $\approx$ 48 min for Model 1 and $\approx$ 120 min for Model 2 respectively.

1.2.3 Inception Score

The Inception Score (IS) [11] gives a measure of the image quality and diversity by evaluating how confidently the Inception model [12] classifies each generated image into a particular class and how diverse the classification results are. It is defined using the Kullback-Leibler (KL) divergence [13]:

$$IS = \exp[\mathbb{E}_x KL(p(y|x)||p(y))], \tag{14}$$

where x are the generated images and y are the predicted classes. A higher score indicates better generation quality and diversity. We train a classifier on 50,000 MNIST images and achieve an accuracy of 0.99 on a test set of size 10,000. We use the classifier on 1000 generated images and calculate the corresponding IS for both models with 10 splits. The results are listed in Table 2. The higher capacity model (Model 2) achieves a higher IS despite the higher noise level. This suggests that Model 2 generates images with higher quality and diversity.

Table 2: Comparison of IS for Model 1 and 2.

Model	Mean	Standard Deviation
Model 1	4.588	±0.152
Model 2	4.946	±0.191

1.2.4 Fréchet Inception Distance

The IS lacks a direct comparison with real images and could thus provide misleading results. A more consistent and reliable metric is the Fréchet Inception Distance (FID) [14] as it directly measures the similarity between the distributions of generated images vs. real images through feature vectors computed using Inception-v3 pretrained on the ImageNet datset [15]. Therefore, FID scores tend to correlate better with human judgments of image quality. The feature vectors are obtained from the last pooling layer and are assumed to follow a multivariate Gaussian distribution. The FID is calculated as the Fréchet distance between these Gaussians [16]:

$$FID = ||\mu_r - \mu_x||^2 + Tr(\Sigma_r + \Sigma_x - 2(\Sigma_r\Sigma_x)^{\frac{1}{2}}), \tag{15}$$

where μ_r, μ_x, Σ_r and Σ_x are the means and the covariance of the real and generated image features. A lower FID indicates are higher similarity between the features, suggesting better generation quality and diversity. We obtain the FID scores in Figure 3 with `torchmetrics` [17] and 1000 generated samples at three different parts of the training process: (i) after 10 epochs, (ii) at the best validation loss epochs (68 and 73), (iii) after the final 75th epoch. The best FID score for Model 1 is 132.1 and 108.9 for Model 2, both achieved at the lowest validation loss epochs.

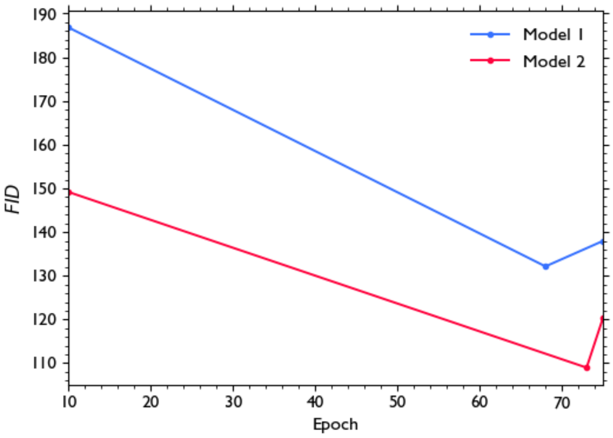
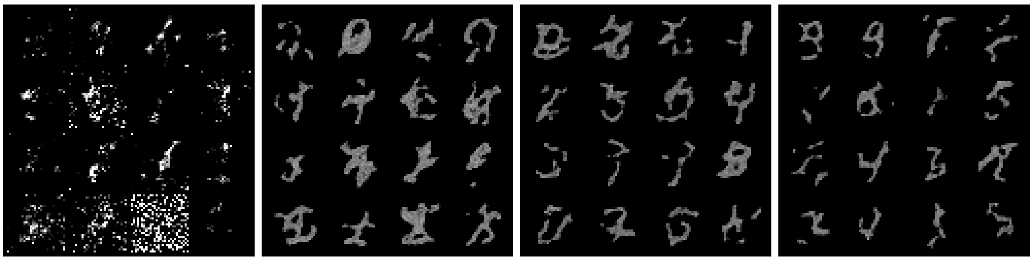


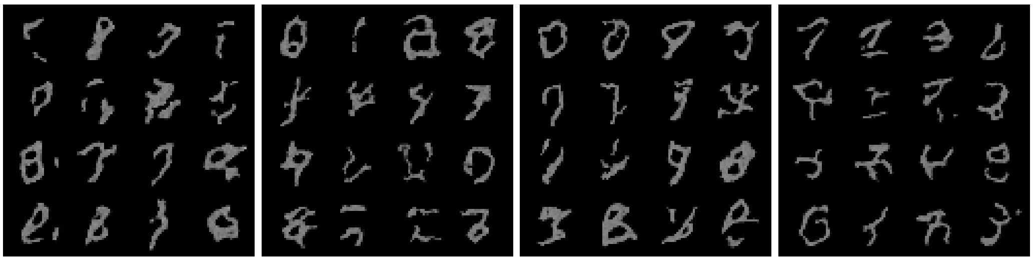
Figure 3: FID scores for Model 1 and 2 at three different points of the training process.

1.2.5 Visual Quality and Diversity

Figures 4 and 5 show a progression of sample quality for both models, suggesting successful parameter updates during training. The lower quality samples during the first epochs correspond to the higher loss and FID values. These show symbols but are barely recognisable as digits for Model 1, while Model 2 generates some digits even at lower epochs. The sample quality and diversity is higher at higher epochs but does not change much after epoch 25, which corresponds to the convergence of the loss curves. Digits are recognisable at higher epochs with Model 2 again showing better quality digits than Model 1. Using the previously trained classifier, we predict the class labels of the generated images and present the distribution in Figure 6. The distribution shows a bias toward the digits 5, 7 and 8 for Model 1, and digits 7, 8 and 9 for Model 2. Overall, the distribution of Model 2 is more balanced, which is in line with the SI and FID results. Note however, that a number of symbols are not actually digits, which makes drawing strong conclusions from the distributions difficult.

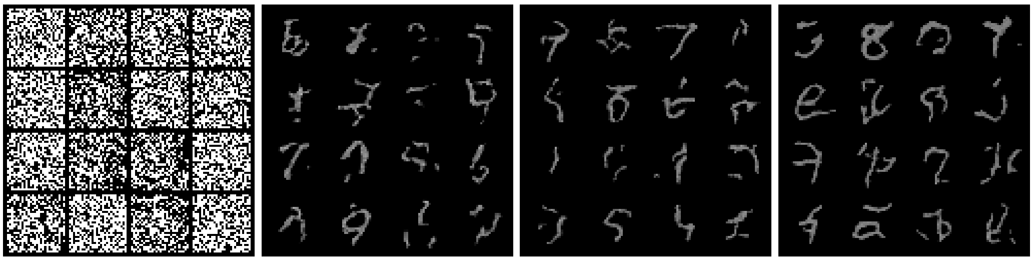


(a) Epoch 1. **(b)** Epoch 5. **(c)** Epoch 10. **(d)** Epoch 15.

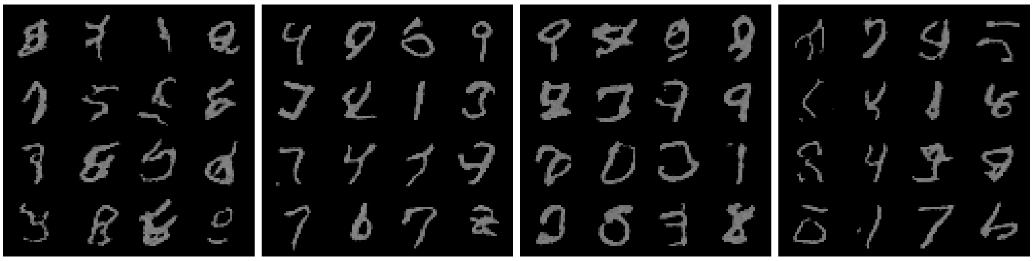


(e) Epoch 25. **(f)** Epoch 50. **(g)** Epoch 68. **(h)** Epoch 75.

Figure 4: Evolution of generated digit images during the training process for Model 1.

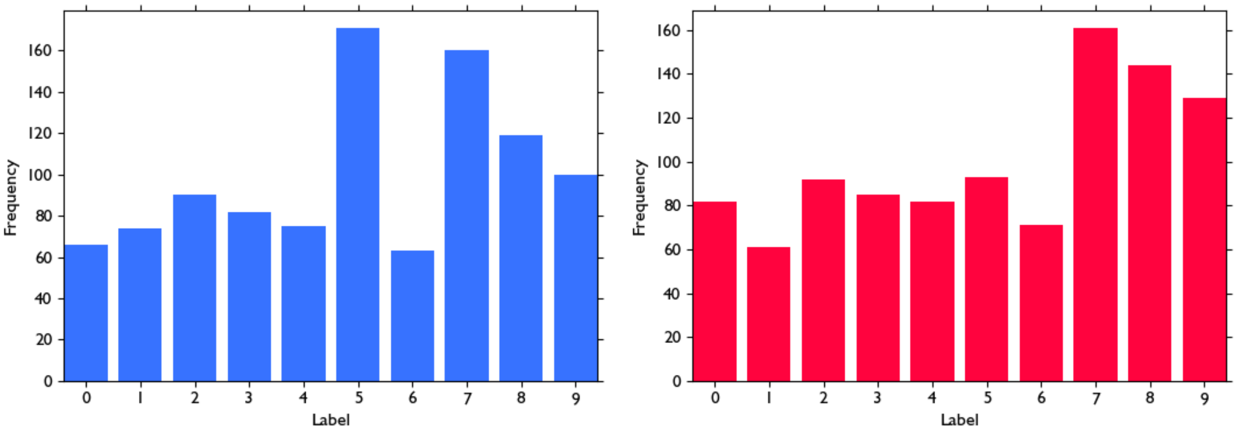


(a) Epoch 1. **(b)** Epoch 5. **(c)** Epoch 10. **(d)** Epoch 15.



(e) Epoch 25. **(f)** Epoch 50. **(g)** Epoch 73. **(h)** Epoch 75.

Figure 5: Evolution of generated digit images during the training process for Model 2.



(a) Training process for Model 1.

(b) Predicted labels for Model 2.

Figure 6: Distribution of predicted digit classes for 1000 generated images from Model 1 and 2.

1.2.6 Analysis and Discussion

The differences in the loss curves, SI, FID and visual results are due to the differences in CNN architectures and hyperparameters as discussed below.

CNN Hyperparameters

CNN Capacity: Model 2 has more hidden units and layers, which means Model 2 is more complex with a higher capacity to learn from the data. More complex models are more prone to overfitting, which may explain the noisier validation loss curves. However, the higher capacity comes with better performance with the minimum loss of Model 2 being half of the one for Model 1, as well as better IS, FID and visual results. The higher complexity of Model 2 also explains the longer training time despite a higher learning rate.

DDPM Hyperparameters

Noise schedule: The noise schedule for Model 2 has a higher final noise level, hence it is trained to recover the data from noisier latents compared to Model 1. This could help the model learn a more robust mapping from the noisy distribution back to the data distribution, enabling better reconstruction and a lower loss. However, the more conservative noise schedule of Model 1 leads to a more stable training process, reflected in the less noisy validation loss curve. The higher noise level for Model 2 may also introduce overfitting risks. **Diffusion Steps:** Model 2 has more diffusion steps, which makes the generative process more gradual and more capable of learning the data distribution, leading to a more fine-tuned model. This is also reflected in the more balanced distribution of generated images.

Training Hyperparameters

Learning Rate: We increased the learning rate for Model 2 due to its higher complexity, to reduce the training time and speed up convergence. However, this also leads to instability in the learning process because higher learning rates may overshoot minima in the loss curve, which explains the fluctuations in the validation curve.

Summary

Model 2 performs better on all performance metrics despite a higher noise level, enabled by its higher capacity architecture and longer diffusion steps. Especially noteworthy are the lower FID score and better visual quality of the generated samples. However, the increased learning rate makes the training process of Model 2 less stable.

2 Custom Degradation

2.1 Part 2(a)

2.1.1 Degradation Strategy

A recent study on *Cold Diffusion* [18] suggests that the performance of diffusion models shows limited sensitivity to the degradation used. The current understanding of these models relies heavily on Langevin dynamics [4], variational inference [19] and Gaussian noise for training and sampling [1, 20]. Instead, we present experimental results using *impulse noise* (or *salt-and-pepper noise*) [21, 22] as a degradation strategy, which introduces sparse and discrete disturbances to the images by adding random white and black pixels to an image.

Impulse Noise

For a given input image I with pixel values in the range $I_{ij} \in [I_{min}, I_{max}]$, the impulse noise transforms the image into I' according to:

$$I'_{ij} = \begin{cases} I_{max} & \text{for} \quad r_{ij} \leq \rho_t \\ I_{min} & \text{for} \quad r_{ij} > 1 - \rho_t \\ I_{ij} & \text{otherwise,} \end{cases} \tag{16}$$

where $r_{ij} \sim \mathcal{U}(0, 1)$ is a random number between 0 and 1 for each pixel and $0 < \rho_t < 0.5$ is the impulse noise density for the t-th out of T time steps in the degradation process. Note that in our case, we work with tensors that are normalised such that $I_{max} = 0.5$ and $I_{min} = -0.5$.

Noise Density Schedule

Instead of a linear schedule for the noise density schedule $\{\rho_t\}_{t=1}^{T}$, we opt for the cosine schedule introduced by [23], because the linear noise schedule in [1] only performs sub-optimally for low resolution images like the MNIST dataset:

$$\rho_t = \cos\left(\frac{t}{T} \cdot \frac{\pi}{2}\right)^2 \cdot (\rho_{start} - \rho_{end}) + \rho_{end}, \tag{17}$$

where $\rho_0 = \rho_{start}$ and $\rho_T = \rho_{end}$ are the boundaries of the noise density schedule. This allows a quasi-linear degradation in the middle of the degradation process, while smoothing the degradation at the extremes $t = 0$ and $t = T$ as shown in Figure 7.

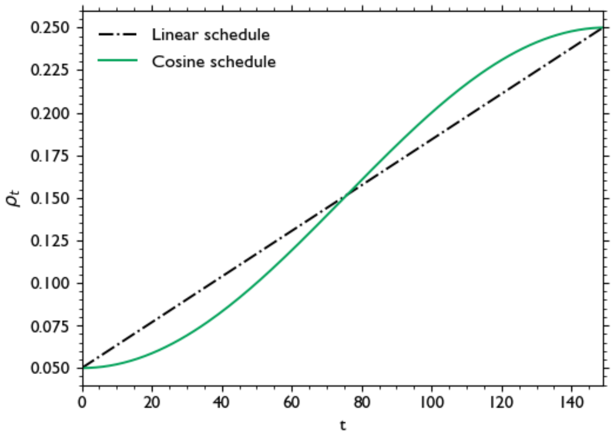


Figure 7: Impulse noise density ρ_t throughout the cosine schedule.

This schedule has been shown very effective for diffusion models, especially when dealing with low resolution images [23]. Moreover, we suggest that it is especially appropriate for aggressive degradations like impulse noise. The impulse noise degradation on an MNIST sample image is visualised in Figure 8 for different ρ_t.

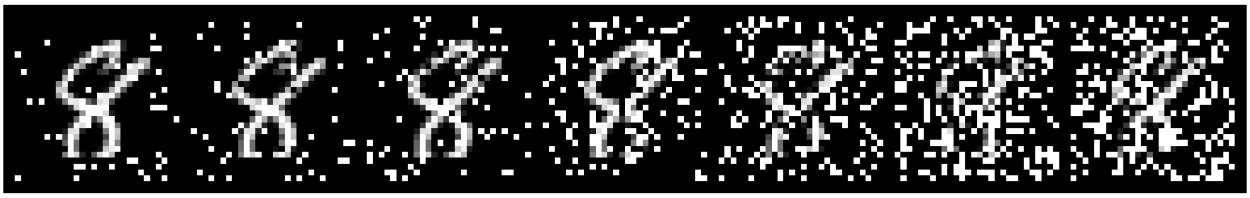


Figure 8: Impulse degradation for $\rho_t = 0.050, 0.063, 0.100, 0.150, 0.200, 0.237$ and 0.250.

2.1.2 Training and Sampling Algorithms

Before, we used a CNN model to predict the noise. With the custom degradation strategy, we instead use a CNN model g_t, also defined as reconstruction operator R, to predict/reconstruct the original image that was degraded using impulse noise, also defined as the degradation operator D. The updated training and sampling algorithms are presented in Algorithms 3 and 4 [18].

Algorithm 3 Diffusion Model Training

```
procedure TrainingDDPM2($p(\mathbf{x}_0)$, $D$, $\rho(t)$)
    repeat
        $\mathbf{x}_{0_i} \sim p(\mathbf{x}_0)$    ▷ Sample from dataset distribution $p(\mathbf{x}_0)$
        $t \sim \text{Uniform}(\{1, \ldots, T\})$    ▷ Sample random time step $t$
        $\mathbf{x}_t = D(\mathbf{x}_{0_i}, \rho_t)$    ▷ Apply degradation $D$ according to noise schedule $\rho(t)$
        $\ell_i = \|g_t\left[\mathbf{x}_t, t/T\right] - \mathbf{x}_{0_i}\|^2$    ▷ Compute individual loss
        Accumulate losses and perform gradient descent to update the model parameters $\phi_t$
    until converged
    return $\phi_t$
```

Algorithm 4 Diffusion Model Sampling

```
procedure SamplingDDPM2($\mathbf{x}_t$, $R$, $D$)
    for $s = t, t-1, \ldots, 1$ do
        $\hat{\mathbf{x}}_0 \leftarrow R(\mathbf{x}_s, s)$    ▷ Restore degraded sample $\mathbf{x}_s$ with model $R$
        $\mathbf{x}_{s-1} = D(\hat{\mathbf{x}}_0, s-1)$    ▷ Reverse degradation with $D$
    end for
    return $\mathbf{x}_0$
```

2.2 Part 2(b)

2.2.1 Model Definition

For the new model (Model 3), we choose a similar CNN architecture as for Model 2, because it achieved better results than Model 1. However, due to poor results, we also introduce regularisation and refer to the regularised model as Model 4.

Regularisation

While Model 3 occasionally produces recognisable digits, we notice that the diversity is low, often producing only one type of symbol as shown in Figure 9. This phenomenon is known as *model collapse* and indicates that the model is not fully capturing the complexity and variability of the target distribution. We also encounter overfitting issues. Therefore, we choose to employ the following regularisation techniques:

1. **Spatial Dropout**: Traditional Dropout randomly drops individual neurons in a layer during training [24]. However, spatial relationships between features are important in a CNN, which is why we instead use Spatial Dropout (`nn.Dropout2d`) [25] with a dropout probability of 0.15. This ensures effective regularisation, while preserving spatial relationships. Moreover, it can prevent the model from over-relying on specific features, thus promoting diversity.
2. **Learning Rate Scheduler:** We also use the `ReduceLROnPlateau` [26] learning rate scheduler to increase the stability of the training process by reducing the learning rate when the validation loss stops improving. We choose a reduction factor of 0.5 with a patience of 3 epochs.

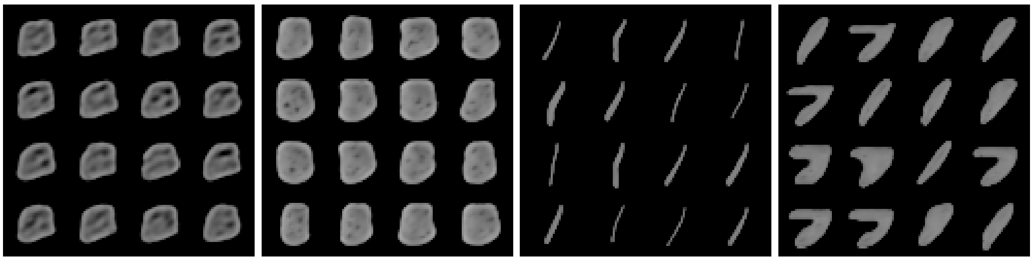


Figure 9: Images at different epochs from training Model 3 displaying very low diversity.

Summary

The hyperparameters for both models are listed in Table 3.

Table 3: Comparison of hyperparameters for Model 3 and 4.

Hyperparameter	Model 3	Model 4
CNN Hyperparameters		
Hidden Units and Layers (`n_hidden`)	(32, 64, 128, 64, 32)	(32, 64, 128, 64, 32)
Activation Function	GELU	GELU
Spatial Dropout Rate	-	0.15
DDPM Hyperparameters		
Noise Schedule (ρ_{start}, ρ_{end})	(0.05, 0.25)	(0.05, 0.25)
Diffusion Steps T (`n_T`)	150	150
Training Hyperparameters		
Initial Learning Rate	2e-4	2e-4
Learning Rate Reduction Factor	-	0.5
Learning Rate Patience	-	3
Optimizer	Adam	Adam

2.2.2 Training and Validation

We train and validate Model 4 with the same dataset as before for 75 epochs. The training and validation loss curves are shown in Figure 10. The gap between training and validation is explained by the regularisation of the model. We choose the weights at epoch 48 as our final Model 4 weights because it produces visually very good results, as shown in Figure 13 on the next page.

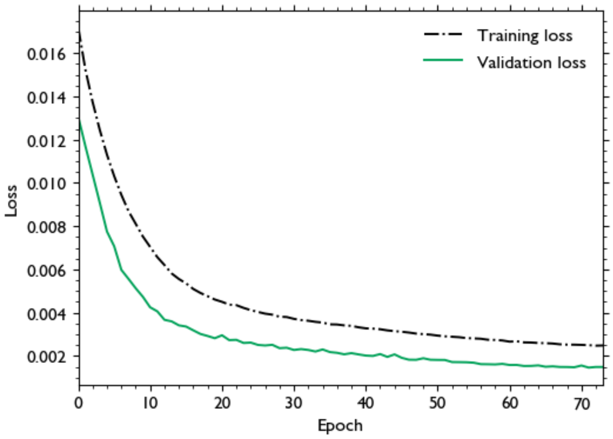


Figure 10: Evolution of training and validation loss with the number of epochs for Model 4.

2.2.3 Image Quality and Diversity Metrics

Inception Score

Using the same method as before, the IS for Model 4 is listed in Table 4. Note that this score is lower than for Model 1 and 2. However, as mentioned before the IS metric is not very robust.

Table 4: IS results for Model 4.

Model	Mean	Standard Deviation
Model 4	3.077	±0.146

Fréchet Inception Distance

We obtain the FID in Figure 11 for Model 4 for three different epochs: (i) after 10 epochs, (ii) at the epoch with the best visual results (48), (iii) after the final 75th epoch. The lowest FID of 75.7 is achieved at epoch 48, which is in line with the visual inspection of the generated images throughout training.

Visual Quality and Diversity

Figure 13 shows the progression of sample quality up to epoch 48. Digits are not recognisable for the first epochs but then achieve high quality in the middle of the training process. A visible decline in quality is observed at later epochs. The variety of digits has greatly increased due to regularisation. However, we observe a shortage of digits 2 and 3, and an abundance of digits 1 and 9 in the predicted class labels distribution in Figure 12.

2.2.4 Analysis and Discussion

After a successful introduction via regularisation techniques, we are able to achieve good results with the impulse noise degradation strategy as shown by the relatively stable validation loss curve, low FID score and very good visual results after sufficient training. The low IS score is likely due to the network bias towards generating certain digits rather than others. However, the IS does not compare the generated images to the real distribution of images, hence any conclusions drawn from the IS are less reliable than the ones from the FID.

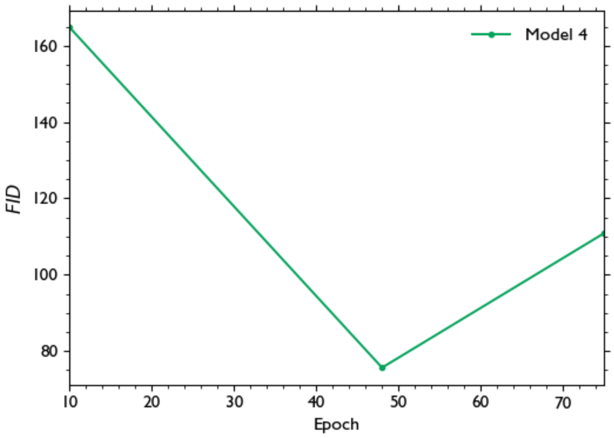


Figure 11: FID scores for Model 4 at three different points of the training process.

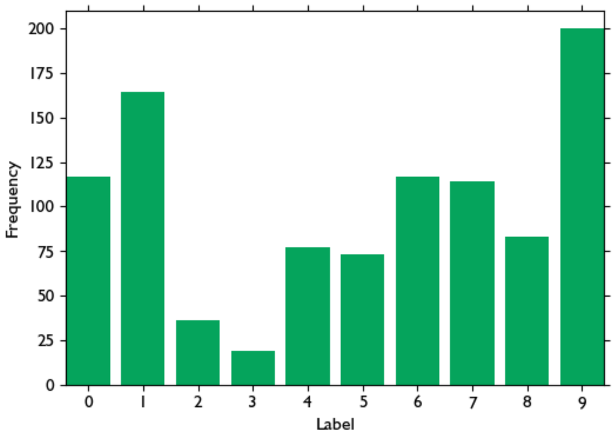
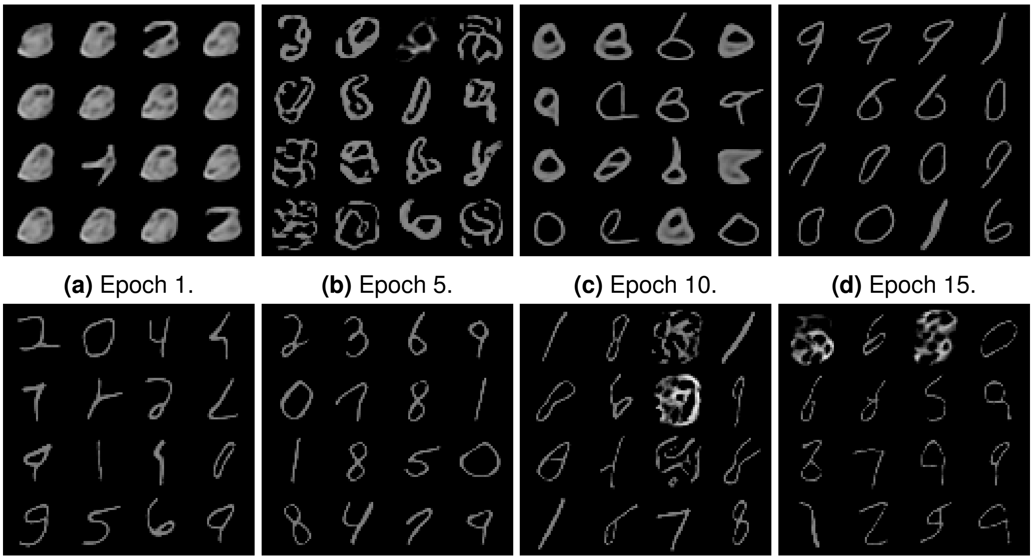


Figure 12: Distribution of predicted digit classes for 1000 generated images for Model 3.



(a) Epoch 1. **(b)** Epoch 5. **(c)** Epoch 10. **(d)** Epoch 15.

(e) Epoch 25. **(f)** Epoch 48. **(g)** Epoch 60. **(h)** Epoch 75.

Figure 13: Evolution of generated digit images during the training process for Model 4.

2.3 Part 2(c)

2.3.1 Gaussian Noise vs Impulse Noise Degradation

We present a comparison between both degradation strategies and focus on Model 2 (Gaussian noise) and Model 4 (Impulse noise) due to their similar CNN architectures.

Loss Curves

Noticeable differences in the loss curves are the fluctuating validation loss for Model 2 and the gap between training and validation loss for Model 4. This is likely due to the introduced regularisation via dropout and a learning rate scheduler. The learning rate for Model 2 is set relatively high.

Inception Score

Table 5 compares the IS results for all models. Model 2 achieves a higher IS score than Model 4. However, this metric is not very reliable as it does not compare the generated images to the real distribution. Drawing the conclusion that the generated images from Model 2 are of higher quality than the ones from Model 4 is not in line with more reliable performance metrics, see below.

Table 5: Comparison of IS for Model 1, 2 and 4.

Model	Mean	Standard Deviation
Model 1	4.588	±0.152
Model 2	4.946	±0.191
Model 4	3.077	±0.146

Fréchet Inception Distance

The evolution of the FID for all models is compared in Figure 14. The FID of the final models is presented in Table 6. We achieve significantly better results for Model 4, despite the more aggressive Impulse noise. However, this is likely due to a significantly higher amount of time spent on fine-tuning the model and introducing regularisation.

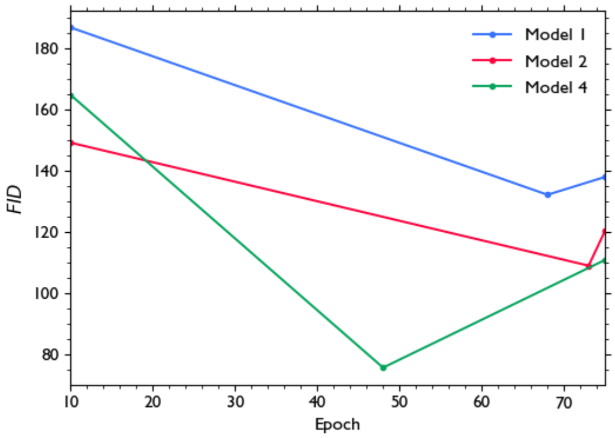


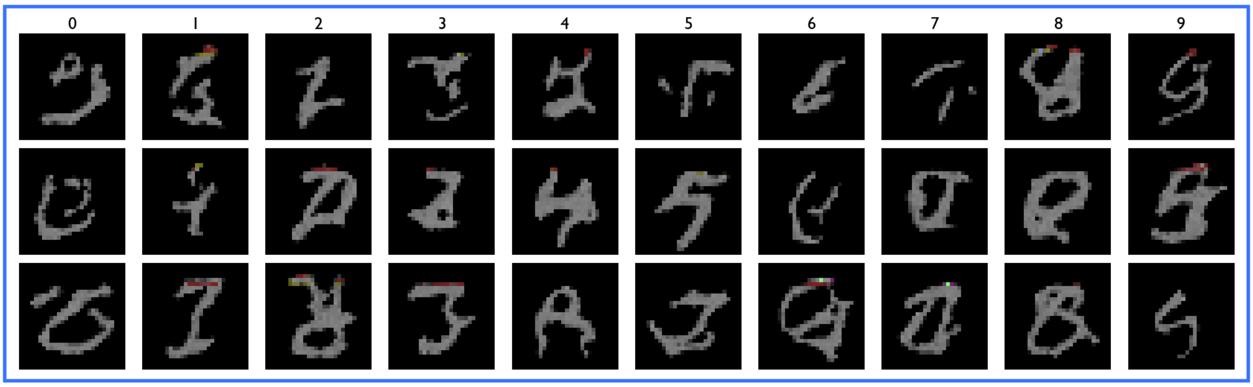
Figure 14: FID scores for Model 1, 2 and 4 at three different points of the training process.

Table 6: Comparison of FID for Model 1, 2 and 4.

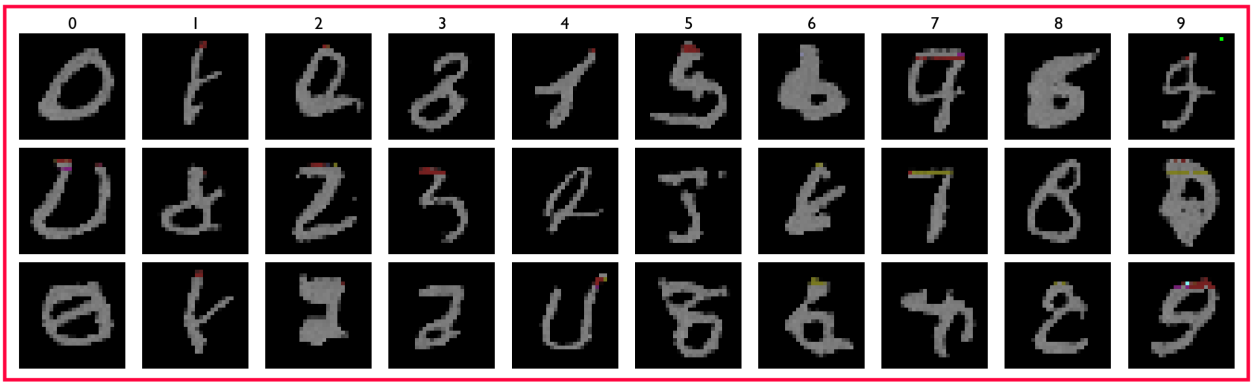
Model	FID	Epoch
Model 1	132.1	68
Model 2	108.9	73
Model 4	75.7	48

Fidelity of Generated Samples

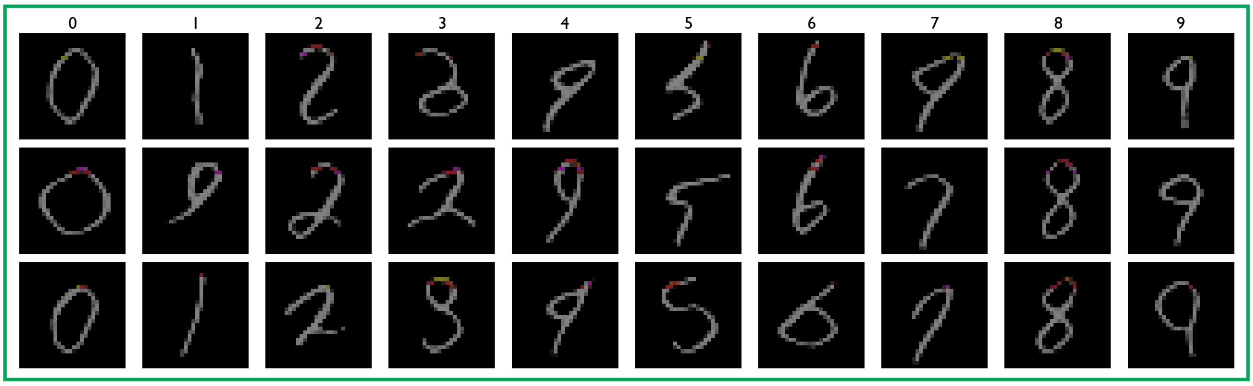
Finally, we compare the fidelity of the generated digits for all models in Figure 15. We can see a progression of the quality of generated images as we go from Model 1 over Model 2 to Model 4. Model 2 shows a few symbols that do not resemble digits, which leads to wrong classification by the MNIST classifier. This might explain the higher IS score for Model 2 compared to Model 4. There are a number of generated digits for Model 2, but they vary in quality with some being thicker than others, and some being more distorted than others. As for Model 4, we observe very high quality digits with uniform line thicknesses and low distortion. However, it seems to have a problem in generating digit 3 accurately, which may be due to its resemblance to digits 8 or 9. This may explain the shortage and abundance of generated digits 3 and 9 respectively in the generated distributions discussed earlier. The higher digit quality for Model 4 is in line with the FID results.



(a) Fidelity of generated samples for Model 1.



(b) Fidelity of generated samples for Model 2.



(c) Fidelity of generated samples for Model 4.

Figure 15: Visual quality of generated digits 0 to 9 for Model 1, 2 and 4 as predicted by the MNIST classifier. Note that the coloured structures in the images are due to an error in saving the images appropriately.

2.3.2 Summary and Conclusions

We have successfully achieved a high generative quality with a diffusion model based on the non-standard and relatively aggressive impulse noise degradation strategy. However, this required a significant amount of experimentation and fine-tuning and is unlikely due to the choice of degradation strategy itself. Initial results of the Impulse noise degradation strategy were significantly worse than the ones obtained from Gaussian noise. We achieved medium-quality results with the Gaussian degradation strategy with practically no hyperparameter fine-tuning. We suspect that we could have achieved better results with the Gaussian noise strategy if enough experimentation was dedicated to fine-tuning the model and potentially introducing regularisation techniques.

Bibliography

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. Advances in neural information processing systems, 33:6840–6851, 2020.

[2] Yann LeCun. The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/, 1998.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32, 2019.

[4] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In International conference on machine learning, pages 2256–2265. PMLR, 2015.

[5] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. arXiv preprint arXiv:2204.06125, 1(2):3, 2022.

[6] Simon JD Prince. Understanding Deep Learning. MIT press, 2023.

[7] Andrey Andreyevich Markov. Extension of the law of large numbers to dependent quantities. Izv. Fiz.-Matem. Obsch. Kazan Univ.(2nd Ser), 15(1):135–156, 1906.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[9] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415, 2016.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[11] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. Advances in neural information processing systems, 29, 2016.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.

[13] Solomon Kullback and Richard A Leibler. On information and sufficiency. The annals of mathematical statistics, 22(1):79–86, 1951.

[14] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems, 30, 2017.

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2818–2826, 2016.

[16] DC Dowson and BV666017 Landau. The fréchet distance between multivariate normal distributions. Journal of multivariate analysis, 12(3):450–455, 1982.

[17] PyTorch Lightning Development Team. TorchMetrics: Machine learning metrics for distributed, scalable pytorch applications. `https://github.com/PyTorchLightning/metrics`, 2023. Accessed: 26 March 2024.

[18] Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise. Advances in Neural Information Processing Systems, 36, 2024.

[19] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. arXiv preprint arXiv:2010.02502, 2020.

[20] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. Advances in neural information processing systems, 32, 2019.

[21] Baoli Shi, Fang Gu, Zhi-Feng Pang, and Yuhua Zeng. Remove the salt and pepper noise based on the high order total variation and the nuclear norm regularization. Applied Mathematics and Computation, 421:126925, 2022.

[22] Keywoong Bae, Suan Lee, and Wookey Lee. Diffusion-c: Unveiling the generative challenges of diffusion models through corrupted data. arXiv preprint arXiv:2312.08843, 2023.

[23] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In International conference on machine learning, pages 8162–8171. PMLR, 2021.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1):1929–1958, 2014.

[25] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 648–656, 2015.

[26] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

A Use of auto-generation tools

A copy of the `README.md` file is attached to this document and can be found on the next pages.

A.1 Co-Pilot

GitHub Co-Pilot assisted the author in producing all function docstrings present in the project repository and a number of descriptive comments in the code. No specific commands have been given, instead auto-completion suggestions have occasionally been accepted.

A.2 ChatGPT

A.2.1 Prompt 1

See README.md for more details.

Prompt:

How do I display 3 versions of digits 0-9?

Output:

```
import matplotlib.pyplot as plt
import os
from PIL import Image

# Your base settings
run_name = "RUN_cnn-nh(32, 64, 128, 64, 32)-actGELU_ddpm-b(0.0001, 0.05)-nt1200_lr0.001"

# Assuming the base path is correct and accessible from this script's directory
base_path_template = f"../contents/{run_name}/predicted_labels{{}}"

# Set up the plot
fig, axes = plt.subplots(nrows=10, ncols=3, figsize=(9, 30))  # 10 digits, 3 versions each
fig.tight_layout(pad=3.0)

for digit in range(10):
    # Construct the path for the current digit
    digit_path = base_path_template.format(digit)

    # Assuming the files are named in a way that sorting them alphabetically orders them correctly
    # or you have some logic to pick three specific files
    image_files = sorted(os.listdir(digit_path))[:3]  # Get the first three images for simplicity

    for i, image_file in enumerate(image_files):
        # Load the image
        img = Image.open(os.path.join(digit_path, image_file))

# Display the image
    ax = axes[digit, i]
    ax.imshow(img, cmap='gray')
    ax.axis('off')  # Hide the axes ticks

plt.show()
```

Modification:

The output was modified by using `matplotlib` instead of `pillow`, added titles for the digits and added a colour box around images:

```
# Display 3 images for each digit

run_name = "RUN_cnn-nh(16, 32, 32, 16)-actGELU_ddpm-b(0.0001, 0.02)-nt1000_lr0.0002(4)"
base_path = f"../contents/{run_name}/predicted_labels"

n_digits = 10
n_images_per_digit = 3

fig, axes = plt.subplots(n_images_per_digit, n_digits, figsize=(n_digits * 2, n_images_per_digit *

for digit in range(n_digits):
    digit_path = os.path.join(base_path, str(digit))
    image_files = sorted(os.listdir(digit_path))[:n_images_per_digit]

    for i, image_file in enumerate(image_files):
        img_path = os.path.join(digit_path, image_file)
        img = mpimg.imread(img_path)

        if i == 0:
            axes[i,digit].set_title(digit, fontsize = 20)

        axes[i, digit].imshow(img)
        axes[i, digit].axis('off')

plt.tight_layout()
rect = patches.Rectangle((0.0, 0.0), 1, 1, linewidth=4, edgecolor=sd2022_green, facecolor='none', t
fig.patches.append(rect)
plt.show()
```

M2 Applications of Machine Learning Coursework Submission (sd2022)

License MIT

Description

This project is associated with the submission of the coursework for the M2 Applications of Machine Learning Module as part of the MPhil in Data Intensive Science at the University of Cambridge. The coursework assignment can be found here: Applications of Machine Learning - Coursework Assignment. The associated report can be found here: Applications of Machine Learning - Coursework Report.

Table of Contents

- Installation and Usage
- Support
- License
- Project Status
- Authors and Acknowledgment

Installation and Usage

To get started with the code associated with the coursework submission, follow these steps:

Requirements

- Python 3.9 or higher installed on your system.
- Conda installed (for managing the Python environment).
- Docker (if using containerisation for deployment).

External Files

The files necessary to run the notebooks in project are found here: M2 Applications of Machine Learning - Submission Files.

The folders in `sd2022\contents` are the following:

- FILES 1: RUN_cnn-nh(16, 32, 32, 16)-actGELU_ddpm-b(0.0001, 0.02)-nt1000_lr0.0002(4)
- FILES FOR MODEL 2: RUN_cnn-nh(32, 64, 128, 64, 32)-actGELU_ddpm-b(0.0001, 0.05)-nt1200_lr0.001
- FILES FOR MODEL 3: SALTPEPPERRAW_cnn-nh(32, 64, 128, 64, 32)-actGELU_ddpm-b(0.05, 0.25)-nt150_lr0.0002 ->
- FILES FOR MODEL 4: SALTPEPPER_cnn-nh(32, 64, 128, 64, 32)-actGELU_ddpm-b(0.05, 0.25)-nt150_lr0.0002(1)

Please simply include it in the `sd2022` repository if you want to run the notebooks successfully.

Additionally, the final models 1, 2, 3, 4 are stored in the Google drive folder `sd2022\ddpm_models`. However, this would require changing the path names in the notebooks to use them directly, and one would not be able to display any information about the training process (loss curves, FIDs at different epochs, sample images at different epochs etc.). Please simply include it in the `sd2022` repository.

Trained classifier models used to predict labels for generated models and calculate the SI scores are stored in the Google drive folder `sd2022\classifier_models`. We recommend using `cnn_classifier4.pth` (default), which has shown the best classification performance on a test set of 10000 MNIST images. Please simply include it in the `sd2022` repository.

Steps

You can either run the code locally using a `conda` environment or with a container using Docker. The Jupyter Notebooks associated with the different parts are located in the `sd2022/src` directory:

- Model 1 : coursework_starter_model1.ipynb
- Model 2 : coursework_starter_model2.ipynb
- Model 3 : coursework_starter_model3.ipynb
- Model 4 : coursework_starter_model4.ipynb
- MNIST classifier : mnist_classifier.ipynb

The Jupyter Notebooks will run faster locally on a high-spec computer (recommended).

Local Setup (Using Conda) [RECOMMENDED]

1. **Clone the Repository:**

   Clone the repository to your local machine with the following command:

   ```
   $ git clone https://gitlab.developers.cam.ac.uk/phy/data-intensive-scie
   ```

   or simply download it from M2 Applications of Machine Learning Coursework (sd2022).

2. **Navigate to the Project Directory:**

   On your local machine, navigate to the project directory with the following command:

   ```
   $ cd /full/path/to/sd2022
   ```

   and replace `/full/path/to/` with the directory on your local machine where the repository lives in.

3. **Setting up the Environment:**

   Set up and activate the `conda` environment with the following command:

   ```
   $ conda env create -f environment.yml
   $ conda activate m2_sd2022_env
   ```

4. **Install ipykernel:**

   To run the notebook cells with `m2_sd2022_env` , install the ipykernel package with the following command:

   ```
   python -m ipykernel install --user --name m2_sd2022_env --display-name
   ```

5. **Open and Run the Notebook:**

   Open the `sd2022` directory with an integrated development environment (IDE), e.g. VSCode or PyCharm, select the kernel associated with the `m2_sd2022_env` environment and run the Jupyter Notebooks (located in the `sd2022/src` directory).

Containerised Setup (Using Docker)

1. **Clone the Repository:**

Clone the repository to your local machine with the following command:

```
$ git clone https://gitlab.developers.cam.ac.uk/phy/data-intensive-scie
```

or simply download it from [M2 Applications of Machine Learning Coursework (sd2022)](#).

2. **Navigate to the Project Directory:**

   On your local machine, navigate to the project directory with the following command:

   ```
   $ cd /full/path/to/sd2022
   ```

   and replace `/full/path/to/` with the directory on your local machine where the repository lives in.

3. **Install and Run Docker:**

   You can install Docker from the official webpage under [Docker Download](#). Once installed, make sure to run the Docker application.

4. **Build the Docker Image:**

   You can build a Docker image with the following command:

   ```
   $ docker build -t [image] .
   ```

   and replace `[image]` with the name of the image you want to build.

5. **Run a Container from the Image:**

   Once the image is built, you can run a container based on this image:

   ```
   $ docker run -p 8888:8888 [image]
   ```

   This command starts a container from the `[image]` image and maps port `8888` of the container to port `8888` on your local machine. The Jupyter Notebook server within the container will be accessible on JupyterLab at http://localhost:8888.

6. **Access and Run the Notebook:**

   After running the container, you'll see logs in the terminal containing a URL with a token. It will look similar to this:

```
http://127.0.0.1:8888/lab?token=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Navigate to http://localhost:8888 and enter the token `xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`. Once you accessed JupyterLab, run the Jupyter Notebooks(located in the `sd2022/src` directory) with an `ipykernel` (Python 3).

Note: Make sure that no other Jupter Notebook Servers are running. Otherwise, you might encounter 'Invalid credentials' issues when entering the token. Close any running Jupter Notebook Servers. To stop a running server, use `Ctrl + C` in the terminal where you launched JupyterLab. Also make sure port `8888` is not occupied.

Support

For any questions, feedback, or assistance, please feel free to reach out via email at sd2022@cam.ac.uk.

License

This project is licensed under the MIT License - see the LICENSE file for details.

Project Status

The project is in a state ready for submission. All essential features have been implemented, and the codebase is stable. Future updates may focus on minor improvements, bug fixes, or optimisations.

Use of auto-generation tools

GitHub Co-Pilot assisted the author in producing all function docstrings present in the project repository. No specific commands have been given, instead auto-completion suggestions have occasionally been accepted. Other uses of auto-generation tools:

ChatGPT Prompt 1: How do I display 3 versions of digits 0-9 if they are stored in these folders: run_name = "RUN_cnn-nh(32, 64, 128, 64, 32)-actGELU_ddpm-b(0.0001, 0.05)-nt1200_lr0.001" base_path = f"../contents/{run_name}/predicted_labels{digit_number}"

- ChatGPT output:

```
import matplotlib.pyplot as plt
import os
from PIL import Image

# Your base settings
run_name = "RUN_cnn-nh(32, 64, 128, 64, 32)-actGELU_ddpm-b(0.0001, 0.05)-nt

# Assuming the base path is correct and accessible from this script's direc
base_path_template = f"../contents/{run_name}/predicted_labels{{}}"

# Set up the plot
fig, axes = plt.subplots(nrows=10, ncols=3, figsize=(9, 30))  # 10 digits,
fig.tight_layout(pad=3.0)

for digit in range(10):
    # Construct the path for the current digit
    digit_path = base_path_template.format(digit)

    # Assuming the files are named in a way that sorting them alphabeticall
    # or you have some logic to pick three specific files
    image_files = sorted(os.listdir(digit_path))[:3]  # Get the first three

    for i, image_file in enumerate(image_files):
        # Load the image
        img = Image.open(os.path.join(digit_path, image_file))

        # Display the image
        ax = axes[digit, i]
        ax.imshow(img, cmap='gray')
        ax.axis('off')  # Hide the axes ticks

plt.show()
```

- Modification of the output:

```
# Display 3 images for each digit

run_name = "RUN_cnn-nh(16, 32, 32, 16)-actGELU_ddpm-b(0.0001, 0.02)-nt1000_
base_path = f"../contents/{run_name}/predicted_labels"

n_digits = 10
n_images_per_digit = 3

fig, axes = plt.subplots(n_images_per_digit, n_digits, figsize=(n_digits *

for digit in range(n_digits):
    digit_path = os.path.join(base_path, str(digit))
    image_files = sorted(os.listdir(digit_path))[:n_images_per_digit]

    for i, image_file in enumerate(image_files):
        img_path = os.path.join(digit_path, image_file)
        img = mpimg.imread(img_path)

        if i == 0:
            axes[i,digit].set_title(digit, fontsize = 20)

        axes[i, digit].imshow(img)
        axes[i, digit].axis('off')

plt.tight_layout()
rect = patches.Rectangle((0.0, 0.0), 1, 1, linewidth=4, edgecolor=sd2022_gr
fig.patches.append(rect)
plt.show()
```

Authors and Acknowledgment

This project is maintained by Steven Dillmann at the University of Cambridge.

28th March 2024