Prepared for submission to DIS-M2 Coursework; Word Count: 2999

Data Intensive Science Applications of Machine Learning Coursework

dw661

University of Cambridge

E-mail: dw661@cam.ac.uk

Contents

1 DDPM Model

1.1 Description of the initial model and training algorithm

We begin by giving a brief explanation of the pre-defined DDPM diffusion model [1] and its training algorithm.

Key definitions and equations for a generic diffusion model:

- Forward Process (Encoder): this process takes an input image and gradually degrades it down to noise. The degredation process is governed by the following equations:

$$\mathbf{z_1} = \sqrt{1-\beta_1}\,\mathbf{x} + \sqrt{\beta_1}\epsilon_\mathbf{1} \tag{1.1}$$

$$\mathbf{z_t} = \sqrt{1-\beta_t}\,\mathbf{z_{t-1}} + \sqrt{\beta_t}\,\epsilon_\mathbf{t} \tag{1.2}$$

  Defining α_t as $\alpha_t = \prod_{i=1}^{t} 1-\beta_i$, the expression for $\mathbf{z_t}$ can be rewritten as:

$$\mathbf{z}_t = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\epsilon \qquad \epsilon \sim N(\mathbf{0}, \mathbf{I})$$

  which allows us to directly sample at time t instead of having to do a full pass. By the endpoint T, $\mathbf{z_T}$ should approximately follow a standard normal distribution.

- Backward Process (Decoder): this process maps samples from the noisy latent space ($\mathbf{z_T}$) back to image space ($\mathbf{x}$). In particular, the probabilitic relationships between the latent variables and $\mathbf{x}$ are characterised as follows:

$$\mathbf{z_T} \sim N(\mathbf{0}, \mathbf{I}) \tag{1.3}$$

$$\mathbf{z_{t-1}}|\mathbf{z_t} \sim N(f(\mathbf{z_t}, \phi_\mathbf{t}), \sigma_t^2) \tag{1.4}$$

$$\mathbf{x}|\mathbf{z_1} \sim N(f(\mathbf{z_1}, \phi_\mathbf{1}), \sigma_1^2) \tag{1.5}$$

  f is a learned neural network that approximates the means of the normal distributions mentioned. Nevertheless, an alternative neural network g is used more in practice; g learns the noises added between each layers as supposed to computing the means.

- The objective function to minimise is:

$$L = \sum_{i=1}^{I}\sum_{t=1}^{T} \left\|\mathbf{g_t}[\sqrt{\alpha_t}\cdot\mathbf{x_i} + \sqrt{1-\alpha_t}\cdot\epsilon_\mathbf{it}] - \epsilon_\mathbf{it}\right\|^2 \tag{1.6}$$

  i.e. the sum of residual squares for predicted noise and actual noise across all T time steps and all I samples.

In the model, the proposed network g consists of convolution blocks with similar architectures. Specifically, in all but the last block, 7×7 convolution kernels are used with padding of 3 and the default stride of 1, on inputs with size 28×28. Under such convolutions, the dimensions of the outputs are identical to that of inputs; after convolutions, `LayerNorm` is implemented followed by `GeLU` activations. The respective channel sizes for these blocks (used in training) are: $16, 32, 32, 16$. In the last block of the network, a convolution with kernel size 3×3 is used with padding 1.

Moreover, temporal information is passed into the network through the block named `time_embed`. The purpose of the block is to provide the network with information on the temporal position of the input.

Specifically, the temporal embedding used in this network is sinuisoidal; the sine and cosine values for each value of $\frac{t}{T}$ is computed (where T is the total number of diffusion steps) and concatenated into a time encoding vector v_t. Then, the outer product between this time vector and a frequency vector is computed, fed into a neural network and the output is added onto the post-activations from the first convolution block. This allows the t to be passed into the network in a nonlinear, high dimensional way, and the parameters of the embedding network are learned during training.

Additionally, the noise schedule parameters β_t are linearly spaced between $[0.0001, 0.02]$ in steps of 2×10^{-5}, which results in 1000 diffusion steps in total. The Adam optimiser was used to train the network with default `beta` values (0.9, 0.99) and a learning rate of 2×10^{-4}.

1.2 Training with different hyperparameters

We experiment with changing the noise schedule β_t. The rationale behind the changes and their impact are analysed in the sections below.

1.2.1 Changing Noise Schedule

From [2], it's mentioned that for low resolution images, linear schedule proposed by the original DDPM paper may be sub-optimal, as the later steps of the diffusion process provide very little contributions since the latents are mostly just noise. One remedy for this is to reduce the amount of added noise at starting epochs. To this end, the 'Cosine Schedule' is proposed; under such schedule, β_t are defined as follows:

$$\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, \qquad \bar{\alpha}_t = \frac{f(t)}{f(0)}, \qquad f(t) = \cos\left(\frac{t/T + s}{1 + s} \cdot \frac{\pi}{2}\right)^2$$

s is an offset parameter to prevent values of β_t being too small near the beginning.

In addition to the Cosine Schedule, we also experimented with two other alternatives: the Constant Schedule and the Inverse Linear Schedule; under Constant Schedule, β_t is chosen to be 0.01 for all t, i.e. the same magnitude of noise is added at every time step. The Inverse Linear Schedule is defined as the flipped version of the original DDPM schedule (i.e. $\beta_t = \beta_{(1000-t)_{DDPM}}$). These two schedules are specifically designed to induce high levels of noise in early epochs, which, in theory, should produce much worse results in comparison to Cosine Schedule and DDPM Schedule as per the arguments from [2].

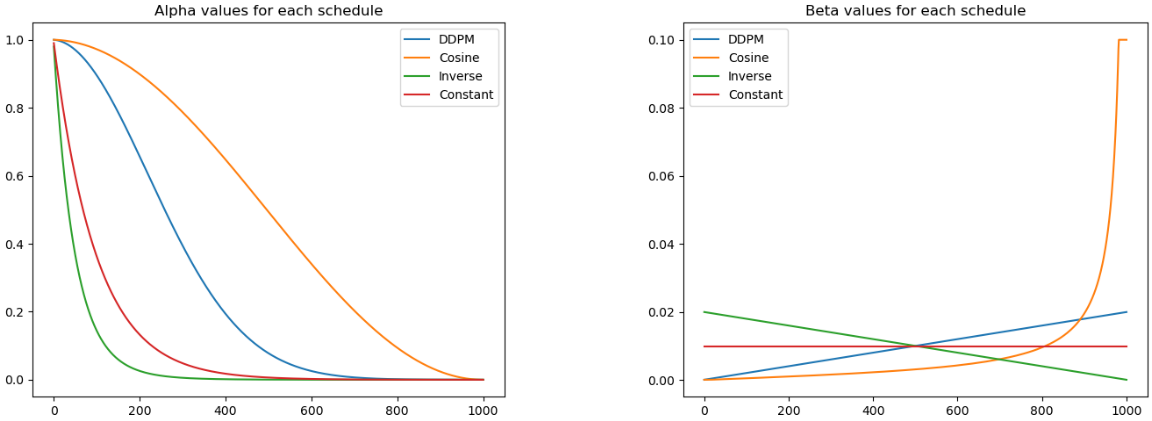


Figure 1: Plots of α_t and β_t values under different schedules.

To further illustrate the effect of noise schedule on the distributions of latent variables $\mathbf{z_t}$, the Kolmogorov-Smirnoff test (K-S Test) [3] is performed on flatten arrays of $\mathbf{z_t}$ at each time step t, to test whether pixel values of degraded images are Gaussian distributed. Using $p = 0.05$ as the cutoff threshold for rejecting the null hypothesis, (values are Gaussian distributed), the following plot illustrates the evolution of p-values of different schedules at varying time steps:

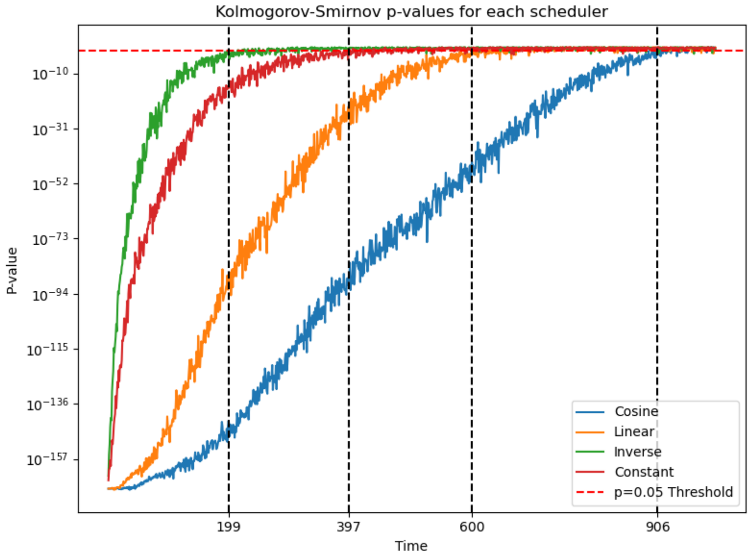


Figure 2: K-S Test values for different schedules at varying time steps

As one can observe, for Cosine Schedule, it takes 906 time steps before pixel values are approximately Gaussian distributed, whereas for other schedules, images degrade into pure noise in substantially fewer steps. This reinforces the previous point that Cosine Schedule provides a much more gradual degradation into noise.

Note that in the implementation of the code, a slight modification is used for Cosine Schedule: instead of clipping the maximum β_t value at 0.999 as the original paper suggested, we clip it at $\beta_t = 0.1$ instead. The reason for that is, through some empirical testing, it can be shown the pixel values tend to explode when β_t is too large near $t = T$ [4], which causes stability issues for training. In practice, clipping at 0.1 has relatively minimal effect on the schedule β_t itself (as only the last 10 out of the 1000 are affected by this clipping) and produces much more stable results.

1.2.2 Documentation of the training process

In this subsection, we document the training process with the different sets of schedulers; the other hyperparameters are fixed at:

- Learning rate: 2×10^{-4}
- β's in Adam Optimiser: Default values as per `pytorch`
- Architecture of network g (same as what's described in 1.1)
- Number of diffused time steps: 1000
- Number of epochs: 50

Below, we illustrate the evolution of the loss curves of the different schedulers during training:

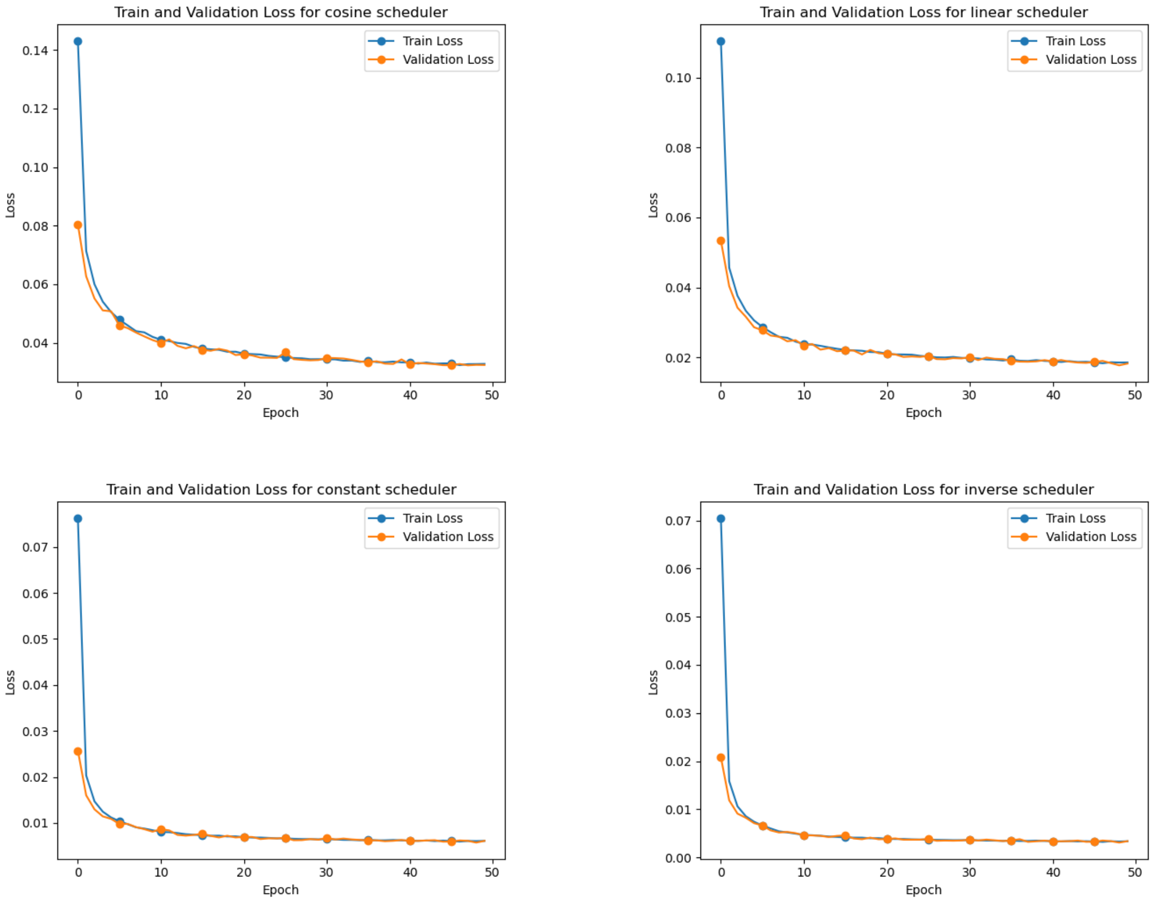
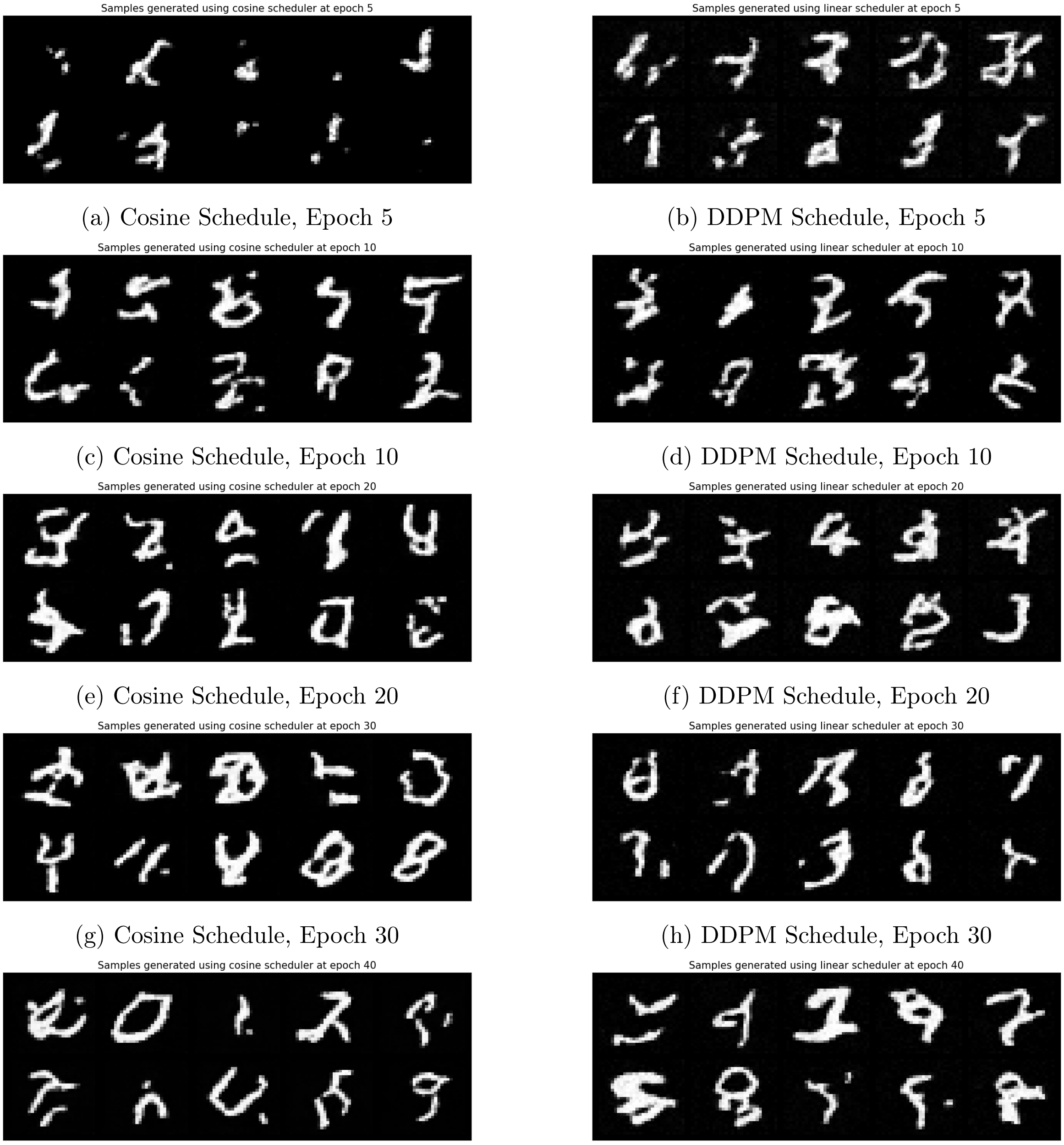


Figure 3: Training and Validation loss against number of epochs

Validation loss was computed on a separate held out test set. For all schedules, the loss curves descend steadily as number of epochs increase. However, it's interesting to note that the magnitudes of losses for DDPM and Cosine schedules are larger than that of Constant and Inverse Linear schedules, which may seem unintuitive at first as they are expected to perform better than the latter two. However, it's crucial to note that the loss being minimised under the DDPM model is the MSE of induced noise at specific timesteps, t, rather than the fidelity of the reconstructed images. Consequently, this metric may not serve as an appropriate basis for comparing the performance of different scheduling approaches.

Instead, quality of generated images at different epochs can be evaluated qualitatively through visual inspection, as well as quantitatively using FID score [5]. FID score uses the pre-trained InceptionV3 model to extract features from real and generated images, fit them to multivariate Gaussian distributions, and compute the Frechet distance between the two Gaussian distributions. A lower FID score indicates closer proximity between the two distributions.

Below, examples of sampled images at different epochs are visualised. Evolution of FID scores under different schedulers during training is also shown:


Samples generated using cosine scheduler at epoch 5
Samples generated using linear scheduler at epoch 5
Samples generated using cosine scheduler at epoch 10
Samples generated using linear scheduler at epoch 10
Samples generated using cosine scheduler at epoch 20
Samples generated using linear scheduler at epoch 20
Samples generated using cosine scheduler at epoch 30
Samples generated using linear scheduler at epoch 30
Samples generated using cosine scheduler at epoch 40
Samples generated using linear scheduler at epoch 40

(a) Cosine Schedule, Epoch 5

(b) DDPM Schedule, Epoch 5

(c) Cosine Schedule, Epoch 10

(d) DDPM Schedule, Epoch 10

(e) Cosine Schedule, Epoch 20

(f) DDPM Schedule, Epoch 20

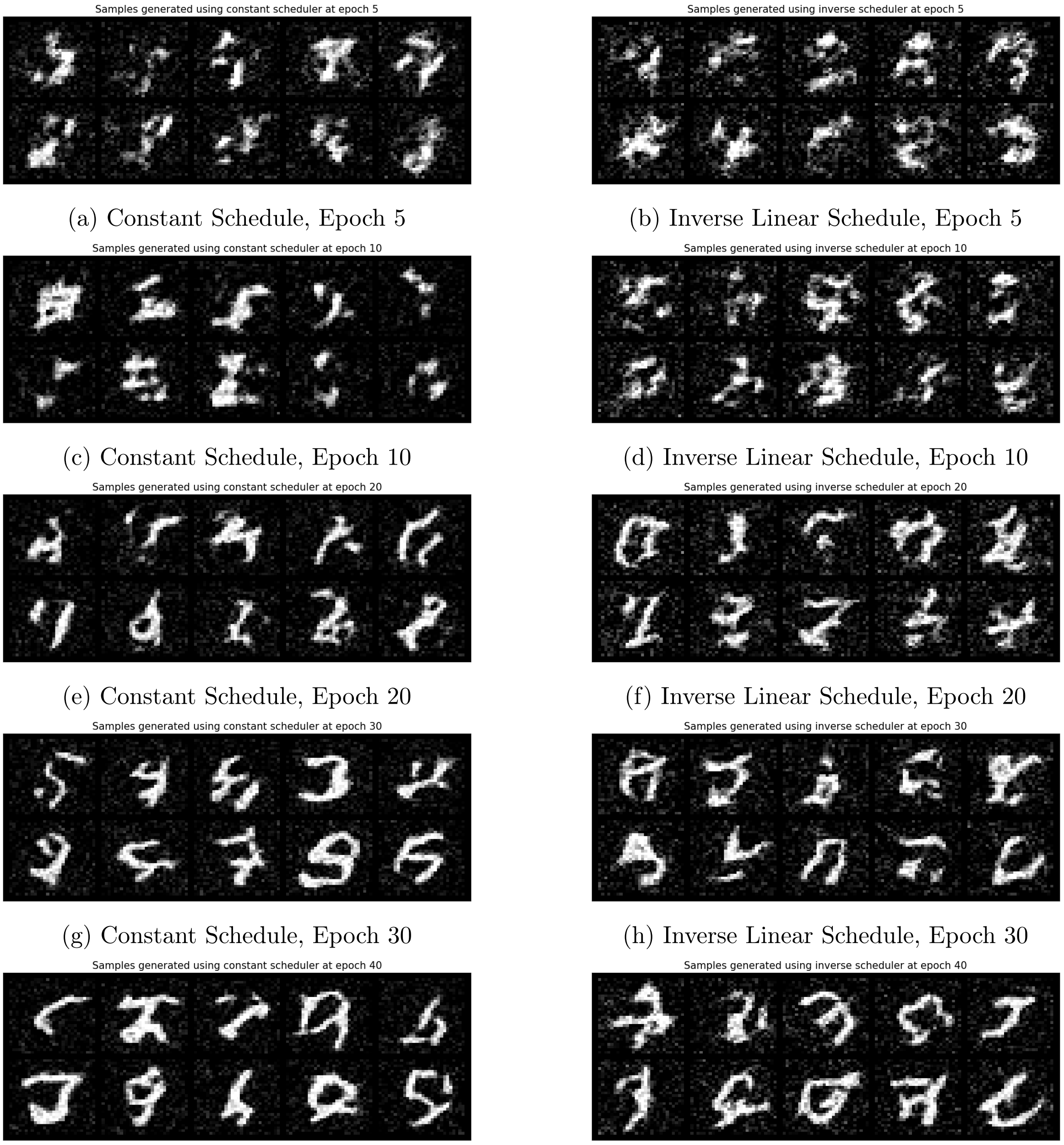
(g) Cosine Schedule, Epoch 30

(h) DDPM Schedule, Epoch 30

(i) Cosine Schedule, Epoch 40

(j) DDPM Schedule, Epoch 40

Figure 4: Generated images at epochs $5, 10, 20, 30, 40$ from DDPM Schedule and Cosine Schedule



(a) Constant Schedule, Epoch 5

(b) Inverse Linear Schedule, Epoch 5

(c) Constant Schedule, Epoch 10

(d) Inverse Linear Schedule, Epoch 10

(e) Constant Schedule, Epoch 20

(f) Inverse Linear Schedule, Epoch 20

(g) Constant Schedule, Epoch 30

(h) Inverse Linear Schedule, Epoch 30

(i) Constant Schedule, Epoch 40

(j) Inverse Linear Schedule, Epoch 40

Figure 5: Generated images at epochs $5, 10, 20, 30, 40$ from Constant Schedule and Inverse Linear Schedule

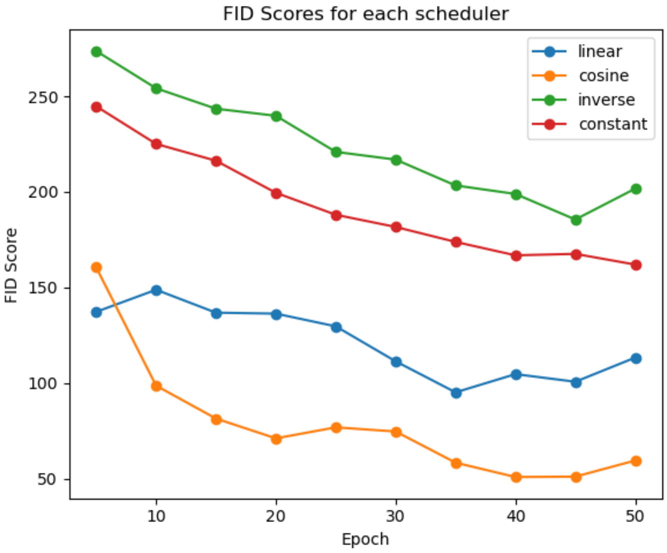


Figure 6: Plot of FID scores for different schedulers at epochs, in steps of 5

Unsurprisingly, Cosine schedule resulted in the lowest FID score. Also, by inspecting the earlier epochs from figures 4 and 5, Cosine and DDPM schedules appear capture the shapes of numbers faster than Constant and Inverse Linear schedules; zoning in on epoch 10 especially, vague outlines of numbers 5, 7 can be seen in the former two, whereas the latter two are still a blur. Nevertheless, in later epochs, outlines of numbers start to appear for all schedulers. The main caveat with the latter schedulers is that they generate noisy images, as indicated by their pixelated backgrounds. This is likely due to the design of the schedulers, as larger magnitudes of noise are added near the end of the decoding process. This may also explain why their FID scores appear to be high in figure 6: it is likely that these noise pixels are being mistakenly interpreted as 'features' in the model, which causes huge disparities when compared to the distribution of features of ground truth images.

1.3 Analysis of the trained models

In this section, we compare the performances of the trained models.

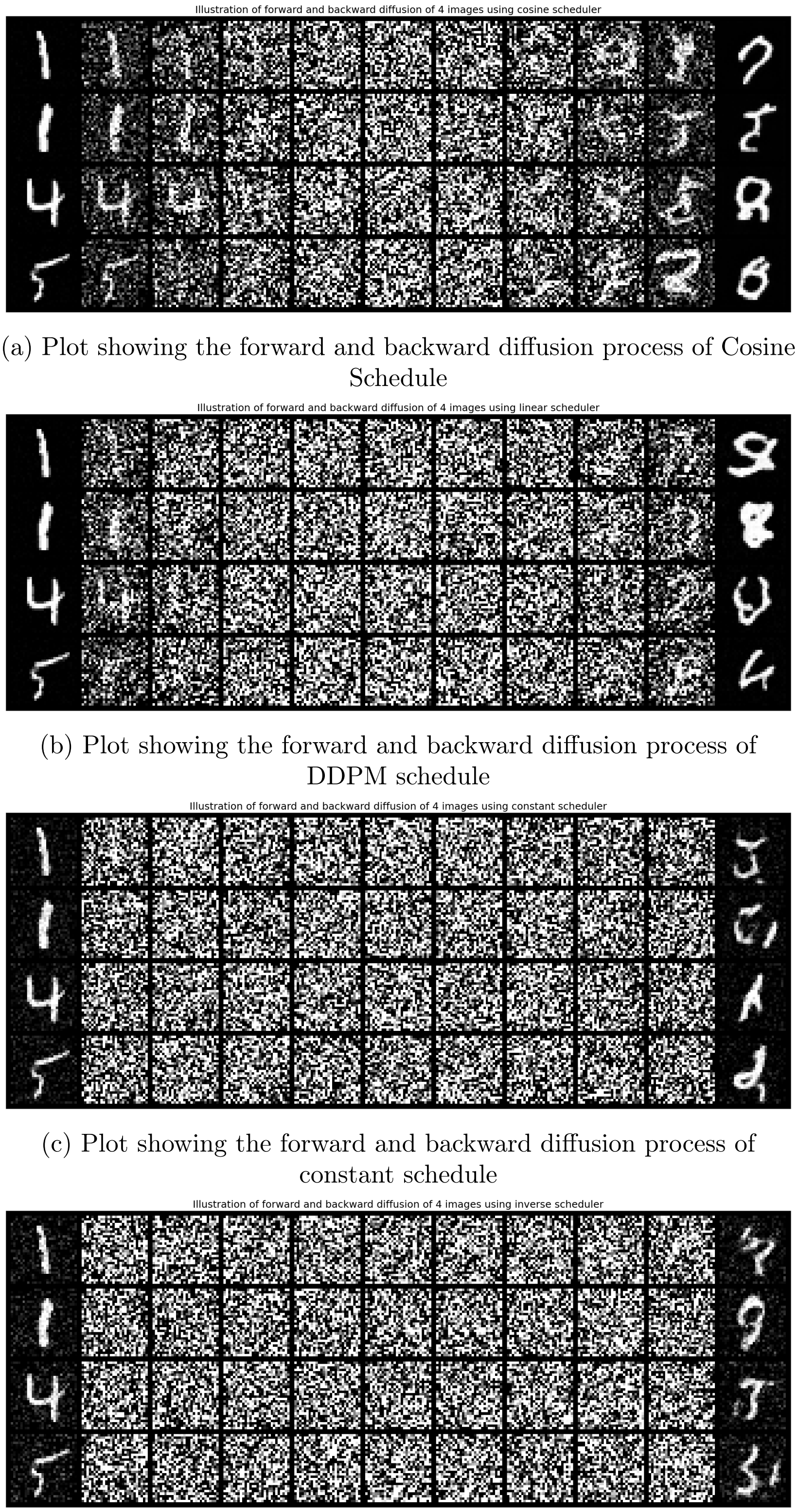


Illustration of forward and backward diffusion of 4 images using cosine scheduler
Illustration of forward and backward diffusion of 4 images using linear scheduler
Illustration of forward and backward diffusion of 4 images using constant scheduler
Illustration of forward and backward diffusion of 4 images using inverse scheduler

(a) Plot showing the forward and backward diffusion process of Cosine Schedule

(b) Plot showing the forward and backward diffusion process of DDPM schedule

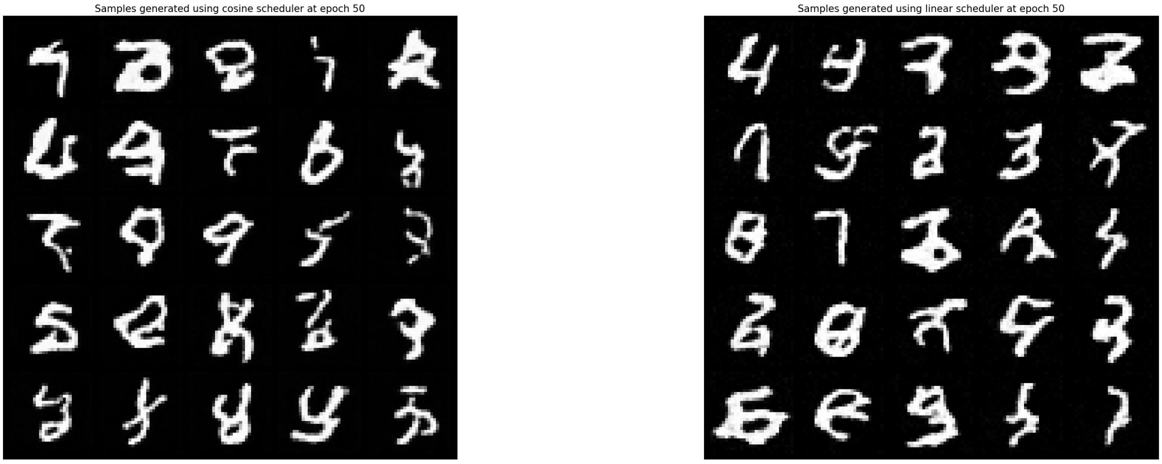
(c) Plot showing the forward and backward diffusion process of constant schedule

(d) Plot showing the forward and backward diffusion process of inverse schedule

Figure 7: Plots showing the forward and backward diffusion processes under different schedules, every 200 time steps

Figure 7 provides indicative insight to the evolution of an input image. In the forward encoding process, almost all information of the input is lost immediately after 200 time steps under Inverse Linear Schedule and Constant Schedule; effectively, these models are just learning mapping from noise to noise after $t = 200$, which makes their 'true' steps of diffusion a lot lower than designed. A similar observation is made during the decoding phase; traces of numerical characteristics didn't reemerge until the process is almost complete, whereas contrarily for Cosine schedule, traces of the numerical features are visibly preserved until the end. This provides empirical evidence for the ideas presented in the paper [2].

Figures 8 and 9 show samples generated directly from latent distributions without conditional input:

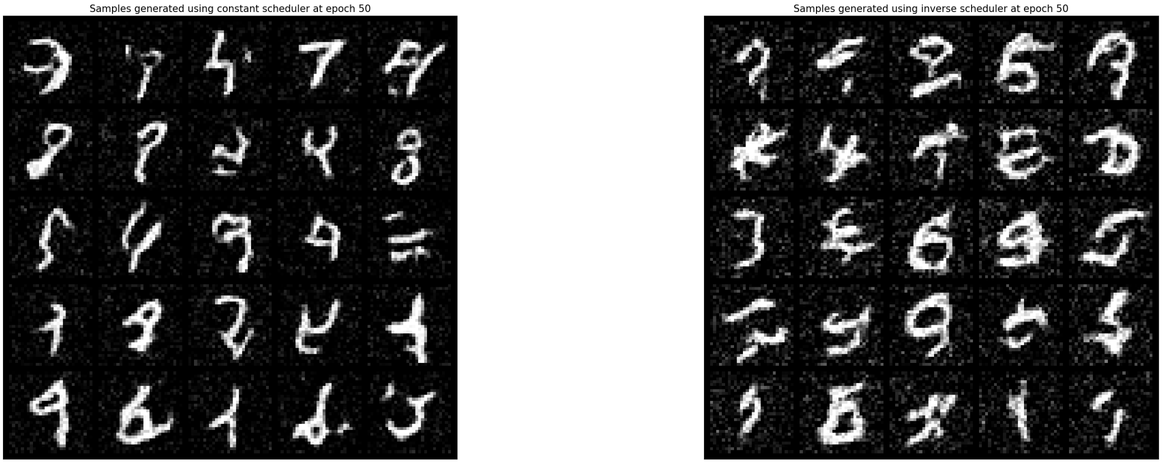

Samples generated using cosine scheduler at epoch 50

Samples generated using linear scheduler at epoch 50

(a) First subplot

(b) Second subplot

Figure 8: Two plots side by side with individual subcaptions.


Samples generated using constant scheduler at epoch 50

Samples generated using inverse scheduler at epoch 50

(a) First subplot

(b) Second subplot

Figure 9: Two plots side by side with individual subcaptions.

Qualitatively, there doesn't appear to be much difference between the linear schedule and

Cosine schedule, even though the difference in FID score at the end of training shown in figure 6 was sizable. Therefore, we conclude that whilst Cosine schedule resulted in marginal improvements in quantitative measures, there isn't sufficient ground to claim that it's superior to the original DDPM schedule.

2 Custom Degradation

2.1 Choice of Degradation

In this section, the alternative degradation strategy of 'morphing' is investigated. This is heavily inspired by [6]. During the forward encoding process, instead of diffusing into Gaussian noise, the images are gradually degraded into random images from the `FashionMNIST` dataset instead. Specifically, `FashionMNIST` was chosen as its images are gray-scale and have the same dimensions as `MNIST`. The restoration process then involves learning the mapping from images in `FashionMNIST` to `MNIST`. The mathematical formulation of the degradation and restoration process can be described as follows:

- Forward Process (Encoder): Using the convention that $\alpha_t = \prod_{i=1}^{t} 1 - \beta_i$, the forward process is described as following:

  $$\mathbf{z_t} = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\mathbf{z} \tag{2.1}$$

  $\mathbf{z}$ is a randomly sampled image from the morphing dataset. By the endpoint T, $\mathbf{z_T}$ should approximately be the same as $\mathbf{z}$. More specifically, following the convention from paper [6], denote the forward process as the 'Degradation operator' $D(x,t) = \mathbf{z}_t$. Similar to section 1.2, we experimented with both linear scheduler and Cosine scheduler for α_t. Their performances are documented and analysed in section 2.3.

- Backward Process (Decoder): define the restoration operator as R; it has the property that $R(\mathbf{z}_t, t) \approx \mathbf{x}$, i.e. it approximates the initial input $\mathbf{x}$ given a diffused input at timestep t. In implementation, R is a neural network, whose parameters are trained by optimising the minimisation problem

  $$\min_{\theta} \frac{1}{N} \|R_\theta(\mathbf{z}_t, t) - \mathbf{x}\|^2 \tag{2.2}$$

  Previously, the latent distribution was approximated to be standard multivariate Gaussian. In the case of morphing, the analogous 'distribution' is just the `FashionMNIST` dataset. When sampling, a random image from `FashionMNIST` dataset is chosen and 'demorphed' back into an `MNIST` digit. In implementation, we use the improved sampling algorithm from [6]:

Algorithm 1 Improved Sampling Algorithm

```
Initialise $\mathbf{z}_T$                    ▷ Fully degraded sample, which is an image from FashionMNIST
for $t = T, T-1, \cdots, 0$ do
    $\hat{\mathbf{x}} \leftarrow R(\mathbf{z}_t, t)$                              ▷ Approximate restoration at $t$
    $\mathbf{z}_{t-1} = \mathbf{z_t} - D(\hat{\mathbf{x}}, t) + D(\hat{\mathbf{x}}, \mathbf{t-1})$                    ▷ Update restoration of $\mathbf{x}$
end for
```

2.2 Neural Network Model

We discuss the changes made to the network structure below. During the implementation of the network, some inspiration were taken from the code repository of Cold Diffusion (in specific, their implmentation of `ConvNext` block). [7]

2.2.1 Time Embedding

The same time embedding method from section 1.1 is still used; however, instead of only adding the encoded time to the first convolution block, it's added to every single convolution block in the network. This change was made to magnify the importance of temporal information.

2.2.2 Network Structure

Instead of the default convolution neural network structure, a modified version of UNet [8] is used. Specifically, it's composed of three downsampling convolution blocks, one middle block, three upsampling convolution blocks and one final convolution block with residual connections between downsampling and upsampling blocks.

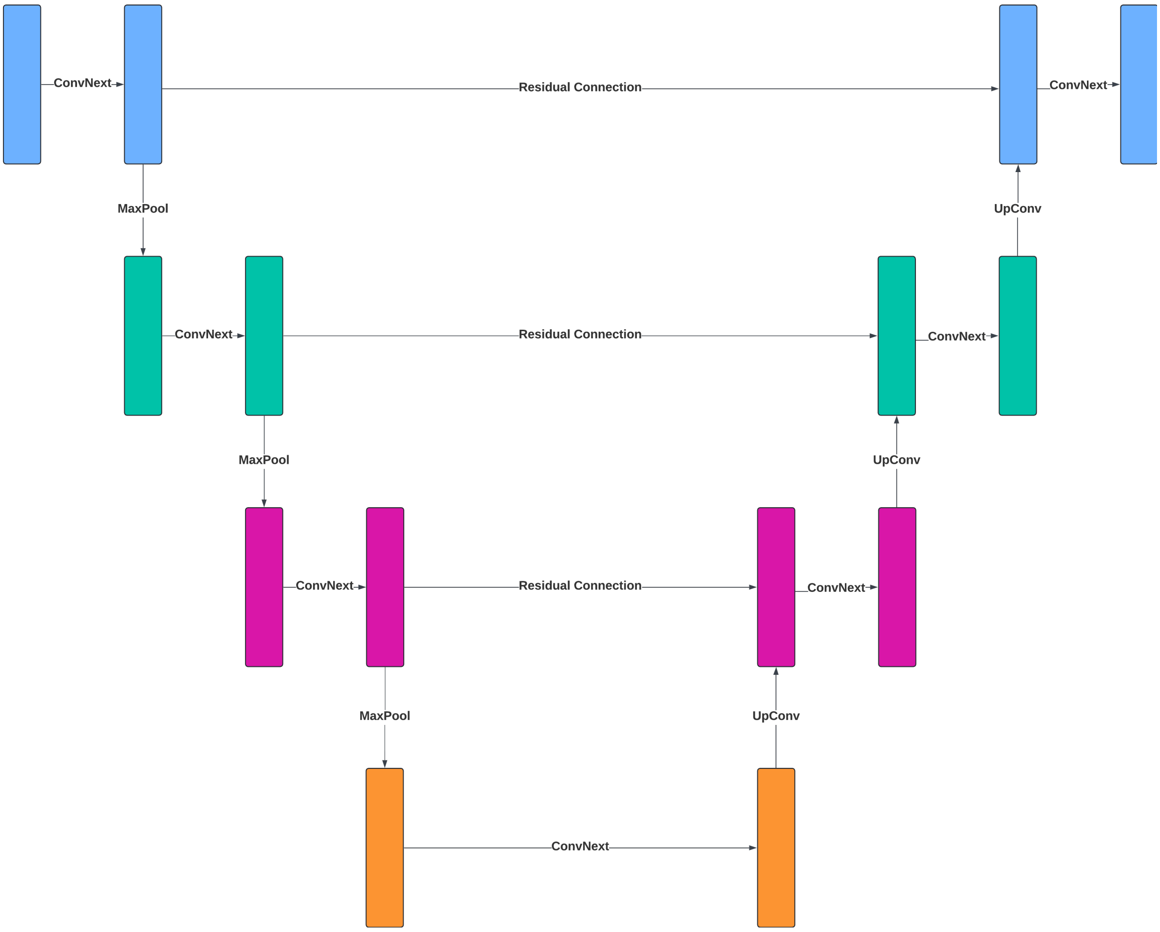


Figure 10: High level architecture of the UNet

- **ConvNext**: ConvNext is a convolution architecture proposed by [9]. In the paper, the authors proposed a convolution block consisting of the following steps:
  - **DepthConv**: Depthwise convolution using a 7×7 kernel.
  - **Main convolution block**: LayerNorm $\to$ 1×1 convolution, with output channels $=$ $4\times$ original input channels $\to$ GELU activation $\to$ 1×1 convolution, with output channels $=$ original input channels

  - **Residual**: The original input x is added back as a residual connection

  In our implementation, slight adaptions were made; before **DepthConv**, a shallow MLP is used to ensure that time encodings passed into the convolution blocks have the same dimensions as the inputs to the block. Also, instead 1×1 convolutions as the paper suggested, 3×3 convolutions were used. Through empirical experimentation with different kernel sizes, it was observed that images generated using 1×1 convolutions had poorer qualities: remnants of images from `FashionMNIST` can often be seen. As for kernel sizes greater than 3, they offered marginal improvements to image qualities, but these improvements weren't large enough to justify for the increase in training time. 3×3 strikes a good balance between the quality of generations and computational efficiency and is thus chosen.

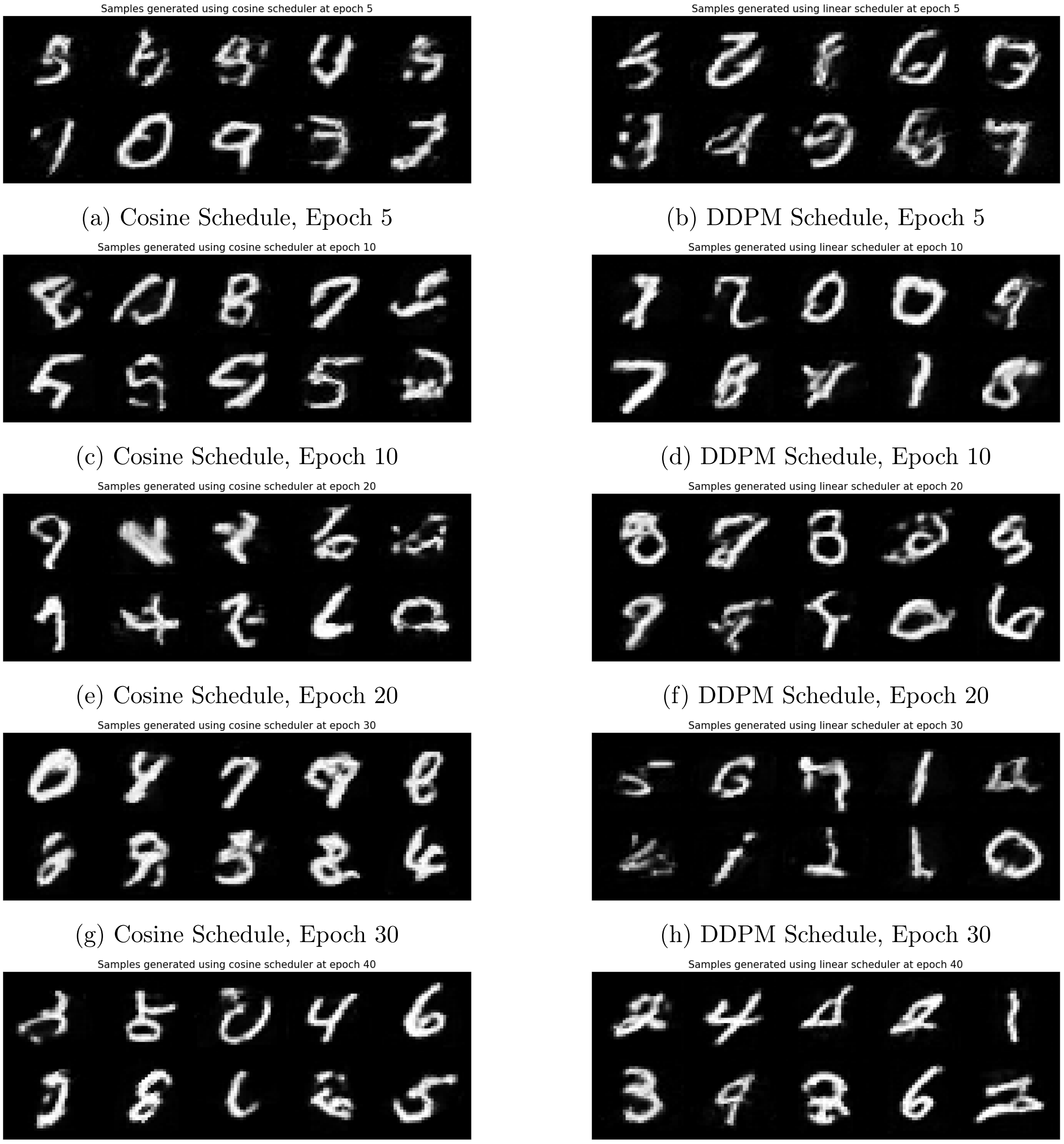
  Lastly, instead of keeping the number of channels between input and output the same as [8] suggested, output channels are doubled during downsampling and halved during upsampling; this change was made to ensure that our network aligns with the overall structure of UNet, whose key idea was having increasing channel sizes during downsampling and decreasing channel sizes during upsampling. The channel sizes of the downsampling and upsampling blocks were chosen to be $32, 64, 128$.

- **MaxPool**: this is a block used to downsample the dimensions of input by a factor $\frac{1}{2}$.
- **UpConv**: this is a block used to upsample the dimensions of input by a factor of 2.

2.3 Documentation of training process and analysis of the end model

2.3.1 Training process

Similar to section 1.2.2, visualisations of generated images at epochs $5, 10, 20, 30, 40$ are presented in figure 11 for linear and Cosine Schedules; in addition, their training and validation loss curves are also visualised in figure 12.


Samples generated using cosine scheduler at epoch 5
Samples generated using linear scheduler at epoch 5
Samples generated using cosine scheduler at epoch 10
Samples generated using linear scheduler at epoch 10
Samples generated using cosine scheduler at epoch 20
Samples generated using linear scheduler at epoch 20
Samples generated using cosine scheduler at epoch 30
Samples generated using linear scheduler at epoch 30
Samples generated using cosine scheduler at epoch 40
Samples generated using linear scheduler at epoch 40

(a) Cosine Schedule, Epoch 5

(b) DDPM Schedule, Epoch 5

(c) Cosine Schedule, Epoch 10

(d) DDPM Schedule, Epoch 10

(e) Cosine Schedule, Epoch 20

(f) DDPM Schedule, Epoch 20

(g) Cosine Schedule, Epoch 30

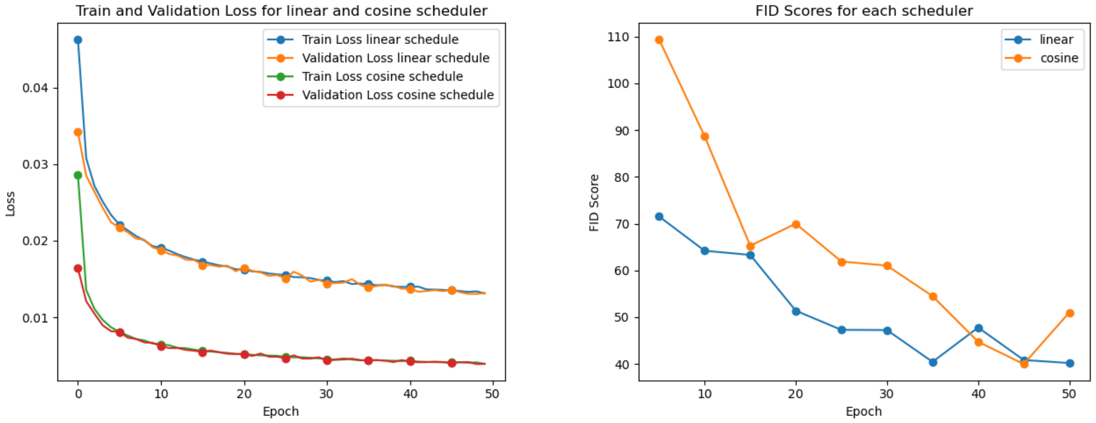
(h) DDPM Schedule, Epoch 30

(i) Cosine Schedule, Epoch 40

(j) DDPM Schedule, Epoch 40

Figure 11: Generated images at epochs $5, 10, 20, 30, 40$ from DDPM Schedule and Cosine Schedule, with morphing as degradation strategy

Visually, there doesn't seem to be huge differences between qualities of generated samples



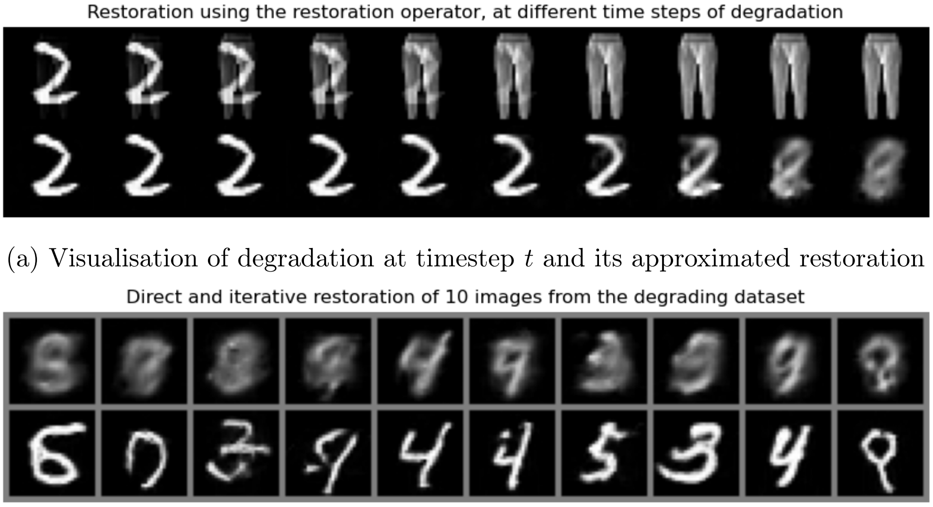
(a) Plot of training and validation loss for both linear and Cosine noise schedule

(b) Plot of FID score for both linear and Cosine noise schedule

Figure 12: Evolution of loss and FID score for both schedulers during training, under custom morphing degradation model

from Linear and Cosine schedules. Therefore, we resort to quantitative measures to decide on which scheduler is superior. Unlike section 1.2.2, the loss characterised by equation 2.2 can be used to assess the generational ability of a model, as it directly calculates the restoration operator's ability to regenerate an image. As can be seen in figure 12, the achieved loss by Cosine Schedule is sizably lower than that of Linear Schedule; moreover, both schedulers achieved similar level of FID score at the end of training. Thus, taking both metrics into account, we take the model trained under Cosine Schedule as the final model.

In figure 13a, a random image $\mathbf{x}$ is chosen from MNIST and degraded to varying time steps t and restored using R; R is the trained restoration operator from the custom morphing model under Cosine schedule. The first row of the plot illustrates the degraded image $\mathbf{z}_t$ at timestep t and the second row illustrates restored image $R(\mathbf{z}_t, t)$. To our delight, the restoration results were high in quality up until timestep ≈ 900, indicating the restoration operator's ability to accurately capture the characteristics of the original image, even when most of its information are lost during degradation. In figure 13b, the comparison of restoring directly using R versus using 1 is visualised. Unsurprisingly, algorithm 1 yielded better reconstruction due to its more sophisticated nature. Nevertheless, the underlying structures of the reconstructed images look very alike. Algorithm 1 effectively deblurred the less pronounced direct reconstructions from R.


Restoration using the restoration operator, at different time steps of degradation
Direct and iterative restoration of 10 images from the degrading dataset

(a) Visualisation of degradation at timestep t and its approximated restoration

(b) Comparison of direct restoration using R and iterative restoration using 1

Figure 13: Plots showing the restoration quality of operator R

2.4 Comparison and Evaluation of the models

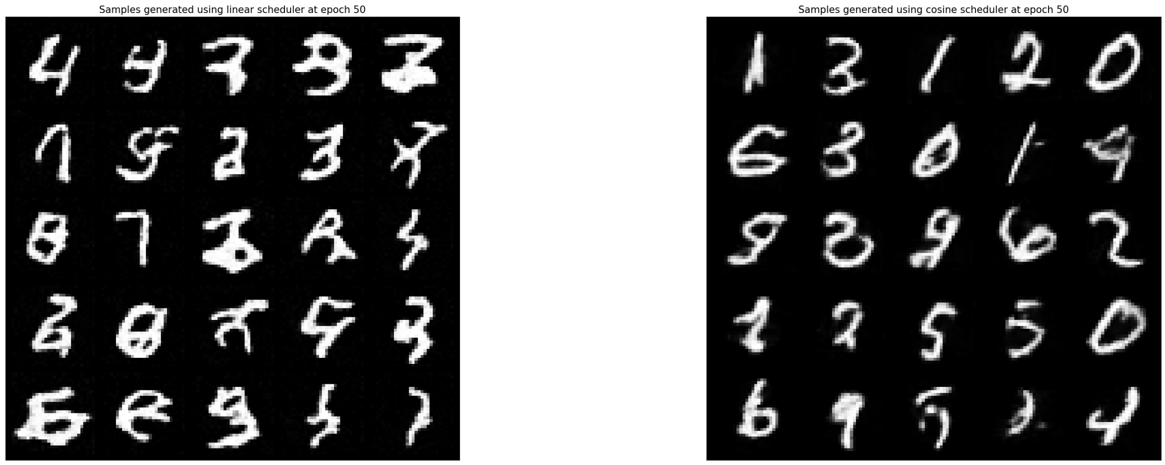
2.4.1 Generated Samples

Figure 14 presents 25 samples generated from the standard Gaussian noise degradation and our custom morphing degradation. From visual inspection, it appears that morphing provided better quality samples. Clear outlines of numbers $0, 2, 3, 5, 6, 7, 8, 9$ can be seen in the latter samples, whereas the default model struggles to precisely identify defining characteristics of individual digits. On a quantitative measure, this improvement in quality is also shown in the FID score. Under the default model, an FID score of 113.31 was achieved at the end of training, whereas the custom morphing model returned an FID score of 50.98, indicating the features of images generated by the latter model aligns more closely than that of the truth dataset.

However, we must be cautious to attribute the success of our custom model to degrading with morphing. In fact, it can be argued that the custom model is only more successful due to its more sophisticated network structure, not because of the choice of degradation. To test this theory, the original DDPM model was trained again, however with UNet as the restoration network instead of CNN. The rest of the hyperparameters were kept the same. Samples from this network are visualised in figure 15. Arguably, these samples are at least the same, if not better, qualities compared to 14b, showing that the superiority of the custom model is maybe more attributable to UNet's ability to capture image characterstics rather than morphing as a degradation strategy.

2.4.2 Pitfall of Morphing

There is also another fundamental detriment to morphing - only a finite number of samples can be generated, as once the model is trained, the mapping from a latent sample to the generated output is deterministic since there is no source of randomness in algorithm 1. The number of unique samples that can be generated is equal to the size of the degrading dataset.


Samples generated using linear scheduler at epoch 50
Samples generated using cosine scheduler at epoch 50

(a) Samples from default DDPM model at epoch 50

(b) Samples from custom morphing model with UNet at epoch 50

Figure 14: Comparison of generated samples

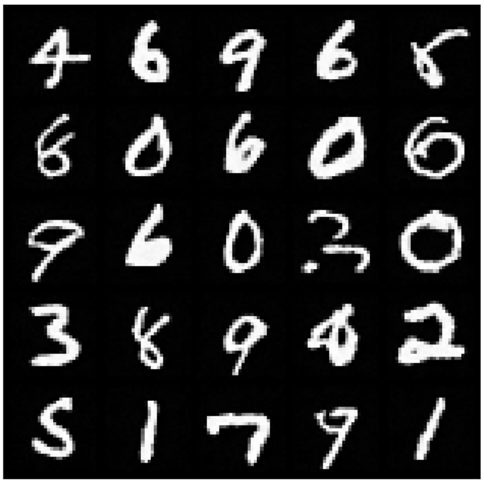
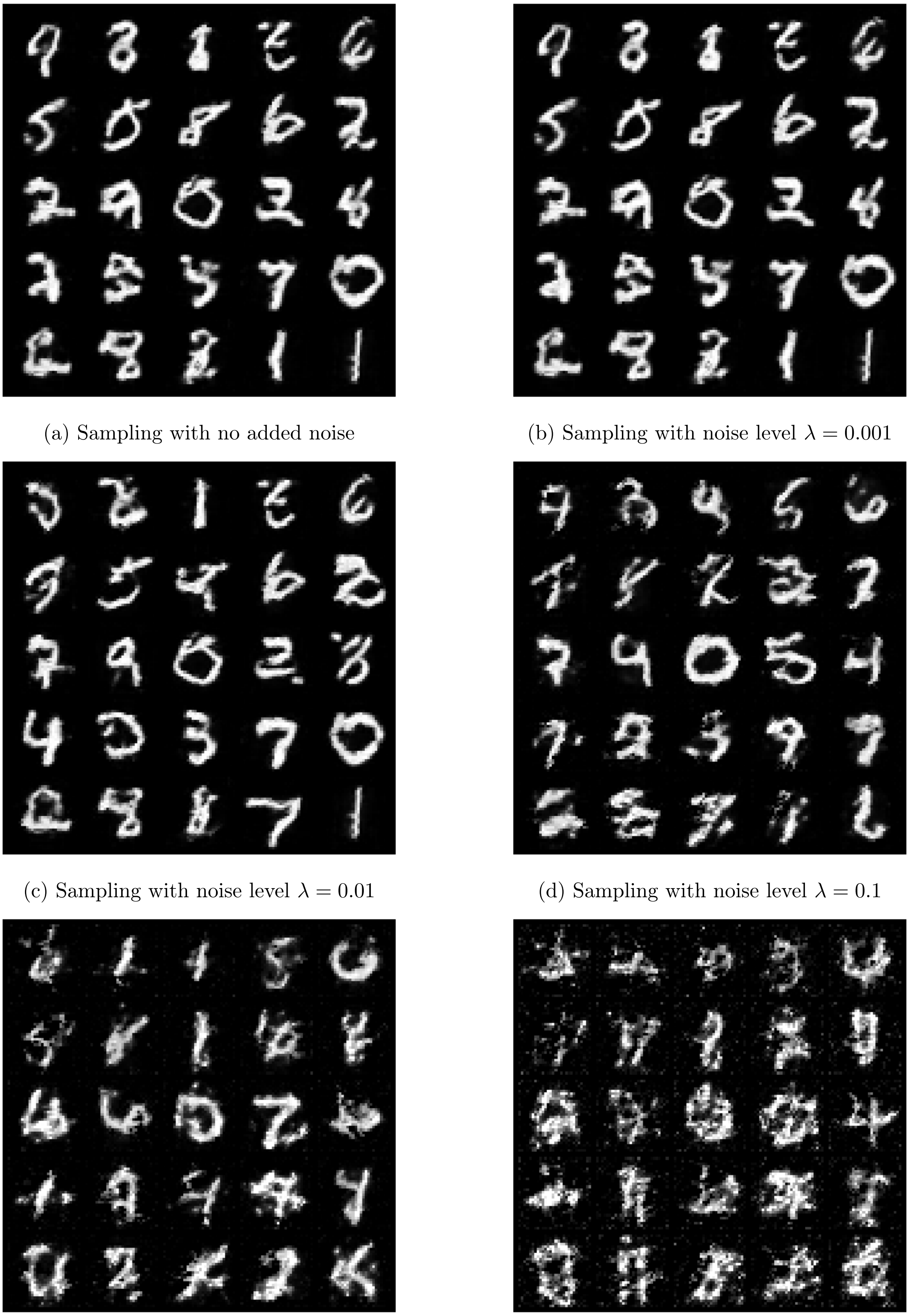


Figure 15: Samples from DDPM model, with UNet as the neural network instead of original CNN

To this end, some attempts were made to overcome this issue. We document these attempts below.

Addition of noise: When sampling, instead of using $\mathbf{z}$, where $\mathbf{z}$ is a random sample from `FashionMNIST`, we introduce a small perturbation $\lambda\boldsymbol{\epsilon}$ and apply algorithm 1 to $\mathbf{z}+\lambda\boldsymbol{\epsilon}$ instead. The distribution of $\boldsymbol{\epsilon}$ is chosen to be $\boldsymbol{\epsilon} \sim N(0, \mathbf{I})$. λ is a parameter which governs the magnitude of induced noise.

During our investigation, added noise levels of $\lambda = 0.001, 0.005, 0.1, 0.25, 0.5$ were experimented. Samples from these attempts are visualised in figure (16), along with generated samples with no added noise:



(a) Sampling with no added noise

(b) Sampling with noise level $\lambda = 0.001$

(c) Sampling with noise level $\lambda = 0.01$

(d) Sampling with noise level $\lambda = 0.1$

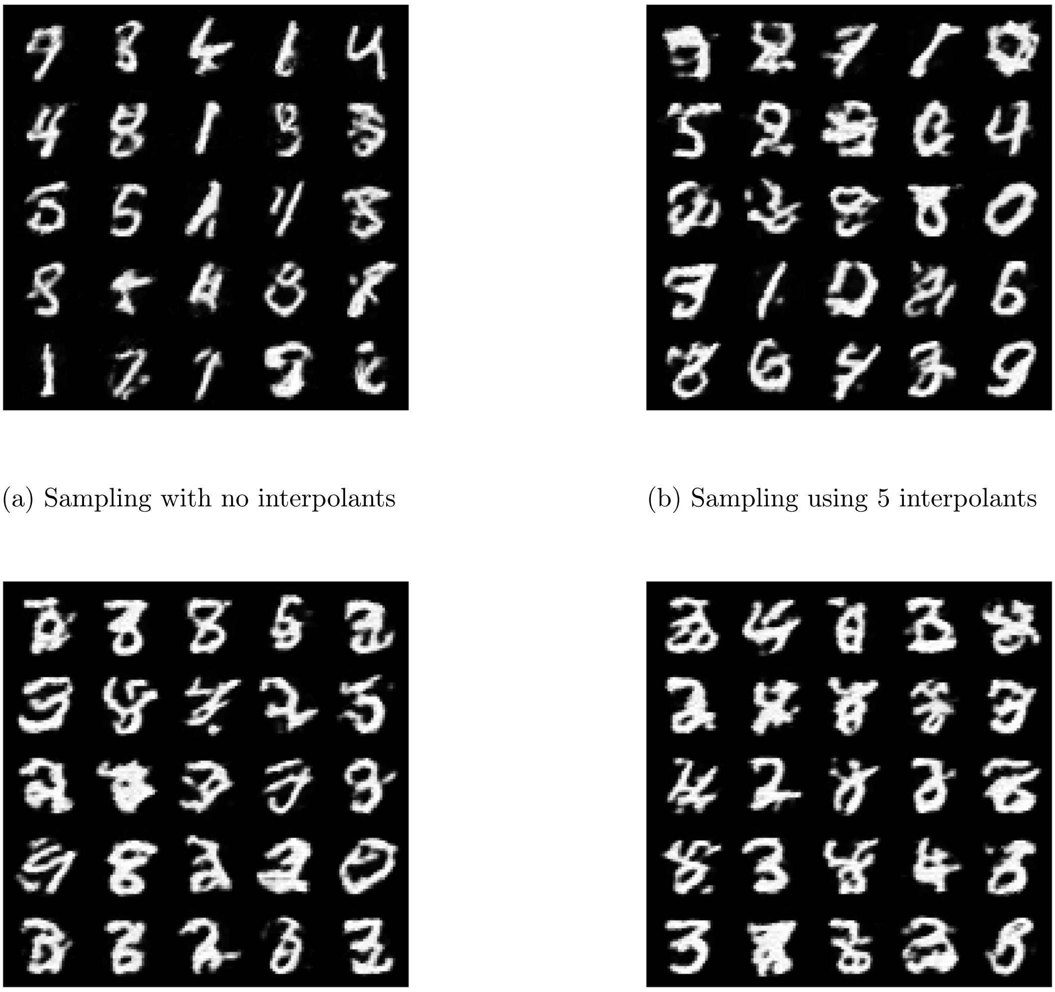
(e) Sampling with noise level $\lambda = 0.25$

(f) Sampling with noise level $\lambda = 0.5$

Figure 16: Generated samples with different levels of added noise

Unfortunately, this attempt had very limited success. As can be observed, when noise level is low, the samples produced are almost identical to the samples produced when no noise is present. On the other hand, when noise level is too high, the generated images become unrecognisable, as they deviate too far from the actual `FashionMNIST` distribution. The best noise level seem to be when $\lambda = 0.01$: a small number of recognisable new samples are generated (for example, row 5, column 4). Nevertheless, there isn't sufficient ground to claim that this is a reliable method of generating new, different samples.

Interpolation: Another strategy investigated was interpolation. Essentially, we first pick N images $\{\mathbf{z}_i\}_{i=1}^{N}$ from the morphing dataset, then combine them into a new sample $z^* = \frac{1}{N}\sum_{i=1}^{N}\mathbf{z}_i$. Since the restoration network is highly nonlinear, we theorise that the output from this won't just be a linear interpolation of the outputs of the individual samples. Some outputs from this are visualised in figure 17.



(a) Sampling with no interpolants

(b) Sampling using 5 interpolants

(c) Sampling using 10 interpolants

(d) Sampling using 20 interpolants

Figure 17: Generated samples using linear interpolations

Unfortunately, similar to adding noise, interpolation also had very limited success. As figure 17 shows, the generated images under interpolation appear smeared and lack clear distinction around the edges. Also, for unknown reasons, generated images showed strong bias towards numbers 3 and 8. This is particularly evident when large number of interpolants are used.

In conclusion, while morphing introduced a new and creative way of image generation, it lacks variety as its latent sampling population is finite and deterministic. One potential way which could remedy this that we haven't experimented with would be fitting a distribution (for example, Gaussian Mixture model) to the latent dataset `FashionMNIST`, and obtain new samples through sampling from this fitted dataset. This would be an interesting area to explore in the future.

References

[1] *Denoising Diffusion Probabilistic Models*, Jonathan Ho, Ajay Jain, Pieter Abbeel, `https://arxiv.org/abs/2006.11239`, Access Date: 08/03/2024.

[2] *Improved Denoising Diffusion Probabilistic Models*, Alex Nichol, Prafulla Dhariwal, `https://arxiv.org/abs/2102.09672`, Access Date: 11/03/2024

[3] *Kolmogorov–Smirnov test*, Wikipedia, `https://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test`, Access Date:17/03/2024

[4] *Pytorch Cosine schedule issue*, `https://github.com/openai/improved-diffusion/issues/105`, Access Date: 15/03/2024

[5] *FID*, Pytorch, `https://pytorch.org/ignite/generated/ignite.metrics.FID.htmle` Access Date: 18/03/2024

[6] *Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise*, Arpit Bansal, `https://arxiv.org/abs/2208.09392` Access Date: 22/03/2024

[7] *Cold Diffusion: Github Repository*, Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie S. Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, Tom Goldstein, `https://github.com/arpitbansal297/Cold-Diffusion-Models` Access Date: 22/03/2024

[8] *U-Net: Convolutional Networks for Biomedical Image Segmentation*, Olaf Ronneberger, Philipp Fischer, Thomas Brox, `https://arxiv.org/abs/1505.04597` Access Date: 22/03/2024

[9] *A ConvNet for the 2020s*, Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, Saining Xie, `https://arxiv.org/abs/2201.03545` Access Date: 22/03/2024

A Use of ChatGPT and Generative AI tools

During the completion of coursework, generative tools such as ChatGPT and CoPilot were used supportively and minimally. All code involving any algorithms or calculations were entirely produced by myself; Copilot was only partially used for Docstring and plotting, and ChatGPT was only used for latex syntax queries. Examples of prompts include:
"How to create a 3 × 3 subplot in Latex?"
"How can I get the first two images from the MNIST dataset?"

B UNet Model

The UNet model used in the second part of the coursework was adapted from a homework assignment for the minor module medical imaging. It wasn't cited since technically I was the author of the model, but original repository of the assignment cam be found at `https://github.com/loressa/DataScience_MPhill_practicals`.