Report for exercise 3 from group C

Tasks addressed:	4
Authors:	Alejandro Hernandez Artiles (03785345) Pavel Sindelar (03785154) Haoxiang Yang (03767758) Jianfeng Yue (03765255) Leonhard Chen (03711258)
Last compiled:	2024–03–02
Source code:	https://github.com/alejandrohdez00/Exercises-MLCMS-Group-C/tree/main/Exercise-3

The work on tasks was divided in the following way:

Alejandro Hernandez Artiles (03785345)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
Pavel Sindelar (03785154) **(Project Lead)**	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
Haoxiang Yang (03767758)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
Jianfeng Yue (03765255)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
Leonhard Chen (03711258)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%

Report on task 1, Principal Component Analysis

In the implementation we do not use the U and S matrices to recover the data but instead recover the US matrix as shown in the theory part of the assignment (by multiplying with V) and then multiply the US by a diagonal matrix to zero the rows corresponding to the low significance singular values we want to remove. This allows us to use both the procedure used in the theory part of the assignment and also the more commonly used method using the data covariance matrix. It also allows us to generalize the transform and reverse transform operations to arbitrary data.

I created two versions of the implementation, one that calculates the V^T using the centered data matrix decomposition and one that calculates it using covariance matrix decomposition. These methods are very similar except for the fact that their U matrices have different shapes and they produce different eigenvalues and eigenvectors.

1.1: How much energy is contained in each of the two components? As we can see in `showcase_PCA.ipynb` roughly 0.99995 percent is contained in the first component and roughly 0.00005 in the second one.

1.2: Plotting the data and adding the direction of the two principal components One thing that caused a measure of confusion was that PCA is usually performed using the SVD of the centered data covariance matrix not the centered data matrix. It seems to be defined like that also in the wikipedia article referenced in the assignment https://en.wikipedia.org/wiki/Principal_component_analysis The covariance (Figure 1) and centered data matrix (Figure 2) do not have the same eigenvectors or eigenvalues but both approaches seems to work.

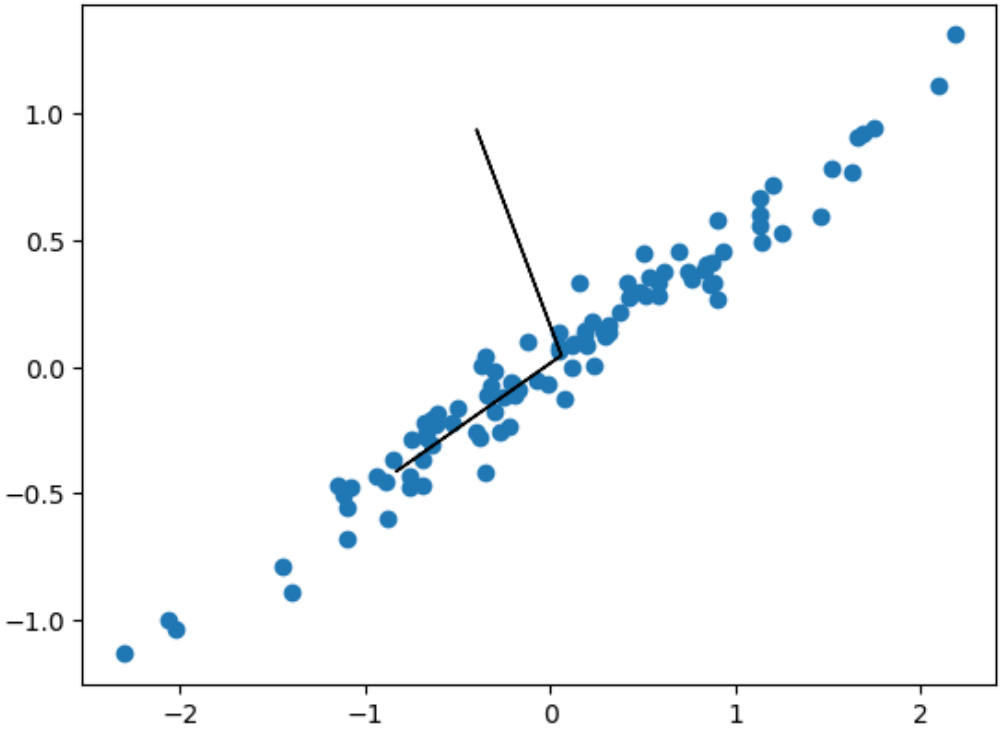


Figure 1: The showcase data set with the direction of the two principal components when using the covariance matrix method

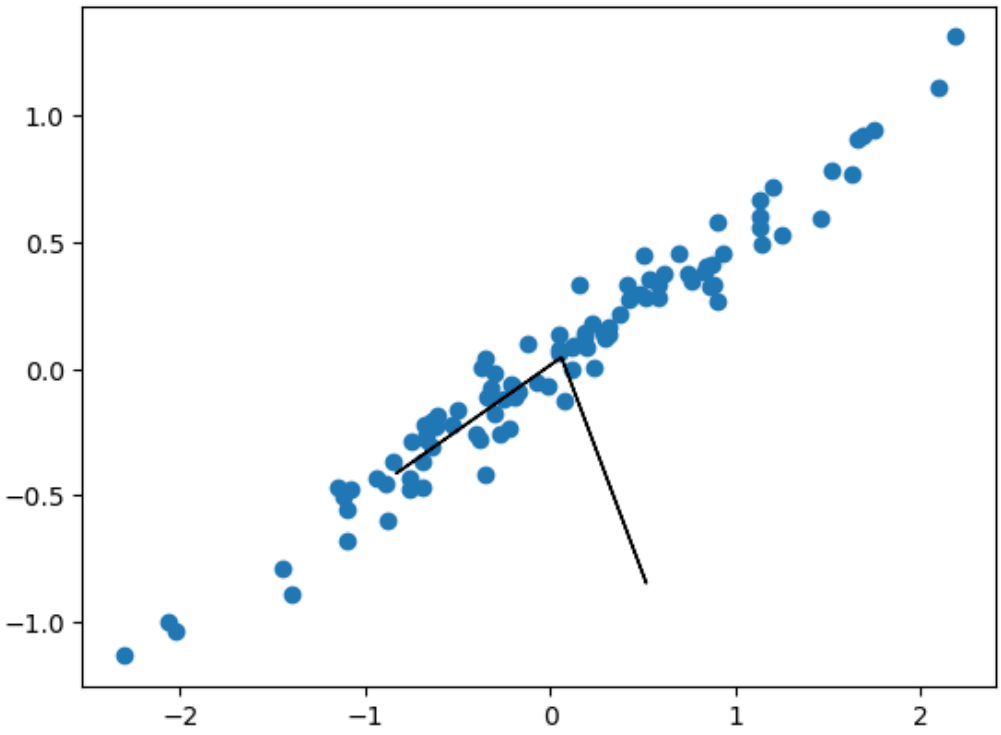


Figure 2: The showcase data set with the direction of the two principal components when using the data matrix method

2.1: Visualizing reconstructions of the image When using many components no change is visible because we are basically just multiplying by something very close to the identity matrix VV^T (Figure 3). When using fewer components the matrix degenerates but the image only seems to degenerate at very low component counts (Figure 4).

When we use the covariance method the results are identical (Figure 5).

(a) all 185 components (b) 120 components

Figure 3: The images restored using the procedure in the exercise definition part 1.

(a) 50 components (b) 10 components

Figure 4: The images restored using the procedure in the exercise definition part 2.

(a) 50 components (b) 10 components

Figure 5: Image restored using the usual covariance matrix method.

2.2: At what number is the information loss visible? The information loss can be clearly seen only when we use 10 or less components (Figure 4). That is because as we will see shortly already at the 9 components the explained variance is more than 99%. Of course since we are only taking into considerations columns and therefore only 1 dimensional neighbourhood and also doing this non-locally the actual information loss is larger.

2.3 : At what number is the energy lost through truncation smaller than 1%? As can be seen in `picture_PCA.ipynb` that would be at nine components.

3.1: Visualize the path of the first two pedestrians in 2D space Can be seen in Figure 6.

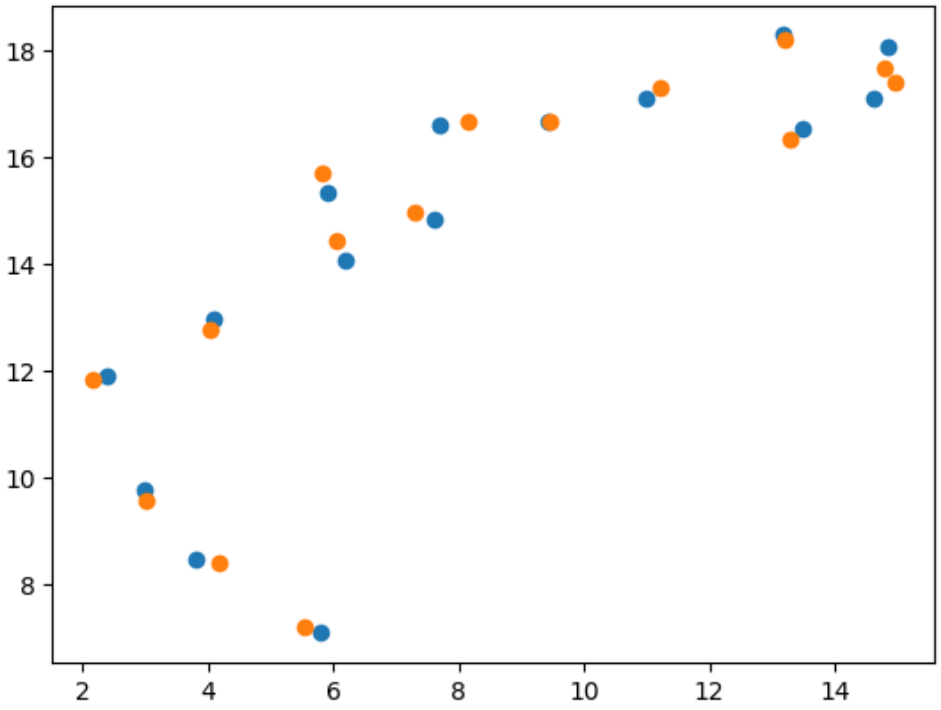
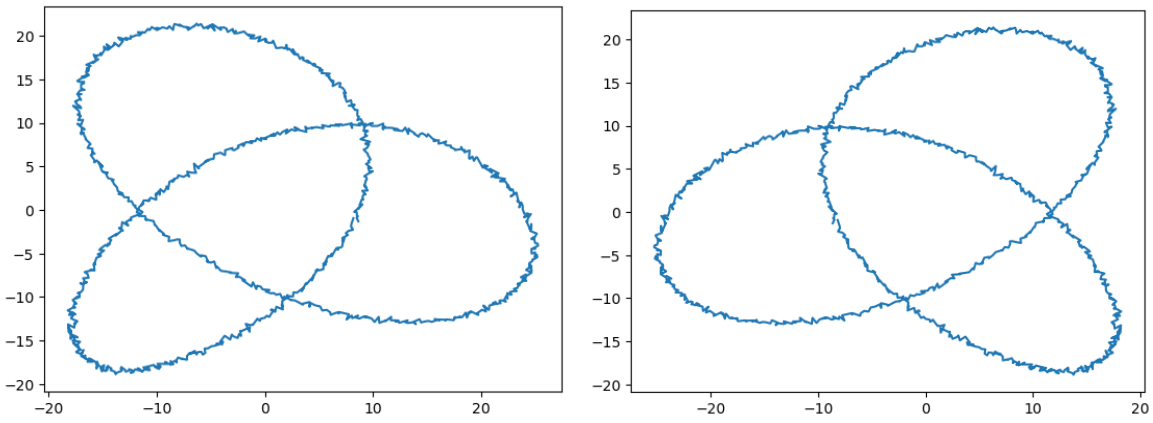


Figure 6: The trajectory of the first two pedestrians

3.2: Are two components enough to capture most of the energy of the data set? Yes as we can see in `vadere_PCA.ipynb` the first components whose cumulative sum is greater than 0.9 are the first two.

Part 3.3: Analyze the data set by projecting the 30-dimensional data points to the first two principal components. Why, or why not? How many do you need to capture most of the energy? When using the centered data matrix the L2 between the same pedestrian in the original and recovered space is 1.244. When using the covariance matrix this is instead 4.529, almost three times as much. The embeddings just look differently oriented (Figure 7).



(a) using covariance matrix

(b) using centered data

Figure 7: Embedding the pedestrian trajectories into 2D space.

The first two components are enough because the trajectories of the pedestrians only vary around the mean trajectory in two dimensions.

Answering the three questions from the first page of the exercise sheet

- (a) An estimate on how long it took you to implement and test the method would be about 5 hours.
- (b) How accurate could we represent the data and what measure of accuracy you used. The average L2 metric between the same pedestrian trajectory in the PCA reconstructed and original data set is about 1.25 (as seen in `vadere_PCA.ipynb`). Since the length of the area in the larger dimension is 30 that seems quite good.
- (c) What we learned about both the data set and the method. The data set seems to contain people walking roughly along some curve. PCA works extremely well for data that is only varying in a few directions around some manifold.

Report on task 2, Diffusion Maps

Implementation For this task, the model was implemented in `diffusion_maps/model.py` and python notebooks for the following tests can be found in the same folder. The `fit(...)` function takes the input data and a `max_dist` for the distance matrix. In the implementation of the diffusion map we purely follow the efficient approach outlined in the exercise sheet. Therefore the `max_dist` argument is for calculating neighbors efficiently using `KDTree` from `scipy`. Also, we used sparse matrices whenever possible to get better performance.

The exercise description recommends using an efficient implementation using sparse matrices and only considering points within a reasonable neighborhood for calculating the distance matrix. During testing it turned out that considering more points leads to better results in general, hence `max_dist` was set to consider all points. Further details will be discussed in each section.

Note that according to [1] $\alpha = 1$ the resulting diffusion map is the Laplace-Beltrami operator. The diffusion map is created according to step 19 in the algorithm from [1] with the timescale fixed to $t = 1$. In our implementation, the ϕ_0 is not included in the diffusion map. Formally we compute our diffusion map embedding using the following expression:

$$\Phi_{\alpha=1,t=1}(x_i) = \left[\lambda_1^t \phi_1(x_i), ..., \lambda_L^t \phi_L(x_i)\right]^T \tag{1}$$

A keen reader would have seen that the exercise sheet outlines the same formula for the Laplace-Beltrami operator using this expression:

$$\Delta\phi_k = \lambda_k \phi_k, \qquad k \in \mathbb{N}$$

The following tests can be reproduced in the python notebooks, see `Exercise-3/README.md` for further clarification.

1.1: Demonstrate similarity between Diffusion Map and Fourier Analysis In the first part we want to demonstrate the similarity between Diffusion Maps and Fourier Analysis. To do this we computed the first 5 eigenfunctions ϕ_l corresponding to the largest eigenvalues λ_l using Diffusion Maps on a data set with $N = 1000$ points. The data set is given by:

$$\begin{aligned} X &= \{x_k \in \mathbb{R}^2\}_{k=1}^N \\ x_k &= (\cos(t_k), \sin(t_k)) \\ t_k &= (2\pi k)/(N+1) \end{aligned}$$

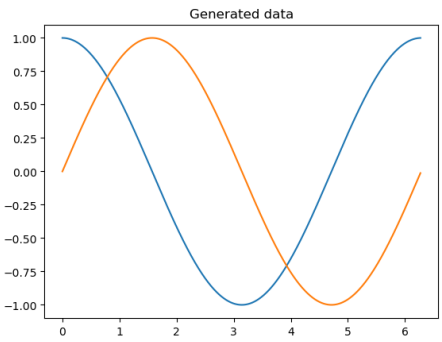
A plot of the first five eigenfunctions ϕ_l can be found in Figures 8 and 9.

The initial dataset is visualized in Figure 8(a). The generated data contains a tuple with coordinates for one period of the *sin*- and *cos*-function. In Figure 8(b) we plotted all eigenfunctions ϕ_l against t_k, while a separate Figure of each eigenfunction (from the diffusion map embedding) can be found in Figures 9. Here we can see ϕ_1 and ϕ_2 reconstruct the *sin*- and *cos*-functions respectively. ϕ_3 and ϕ_4 contain 2 periods of the *sin*- and *cos*-function, but shifted by a phase of $period/2$. We can observe in ϕ_5 3 periods of the *sin*-function (with a phase-shift of $period/2$) and assume that with increasing l of the eigenfunctions ϕ_l more periods are squeezed into the window given by the t_k.

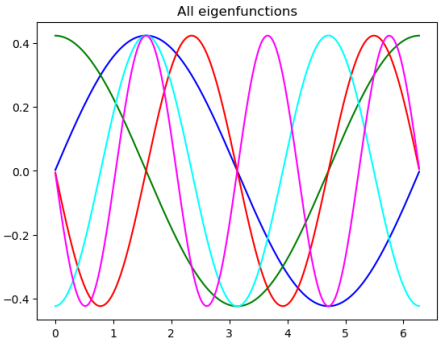
Before comparing the eigenfunctions ϕ_l with the Diffusion Map coordinates, it is important to note that running the experiment can result in phase shifts of $period/2$ for each ϕ_l. So for example ϕ_1 was sometimes the $-sin()$-function. Secondly, the initial testing was done with `max_dist=1.0` for the distance matrix. This led to the calculation an embedding with *sin*- and *cos*-functions of higher frequencies. All presented results were calculated with `max_dist=9.0` which takes all points into account for calculating the distance matrix.

Next in Figure 8(d) the diffusion map coordinates (see 1) were plotted against t_k. We can observe that between Figures 8 (b) and (d), that amplitudes are scaled down by the corresponding eigenvalues λ_l. We conclude the eigenvalues λ_l could describe the importance of each eigenfunction ϕ_l. In the following section whenever we mention the eigenfunction ϕ_l it refers to the diffusion map embedding (see 1).

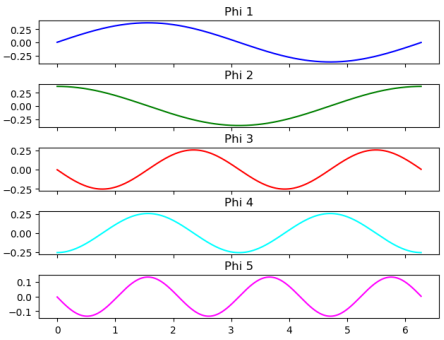
Further observations of ϕ_l for large l leads to the assumption that ϕ_l for $l \in \mathbb{N}$ represents the *sin*- and *cos*-functions correspondingly at higher frequencies (disregarding the phase-shift of $period/2$). It is also notable when `max_dist` is low that a sparser distance matrix causes an increase of the initial frequency observed in ϕ_1.



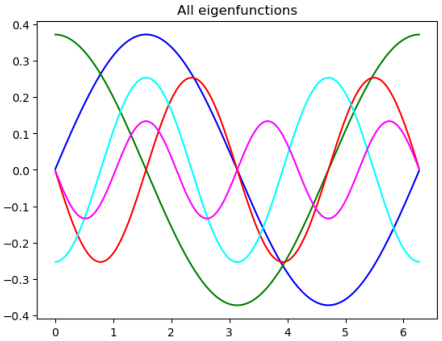
(a) Fourier data set



(b) All eigenfunctions ϕ_l

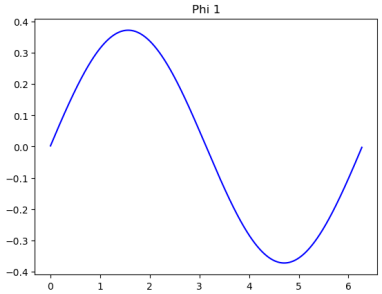


(c) Comparison of diffusion map functions

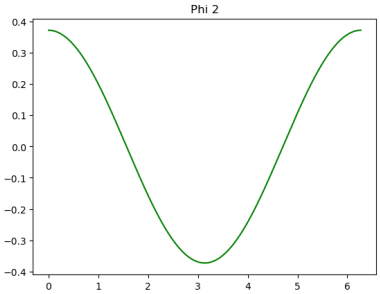


(d) All diffusion map functions

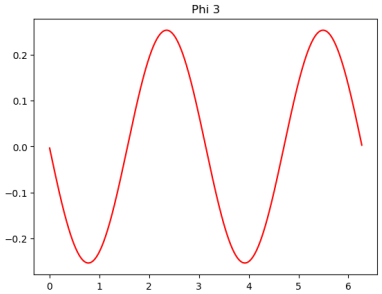
Figure 8: Diffusion Map results



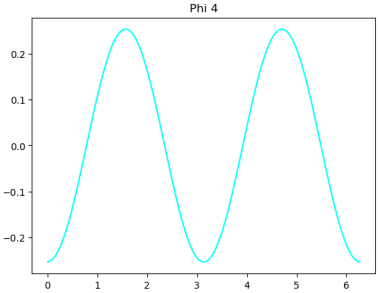
(a) ϕ_1



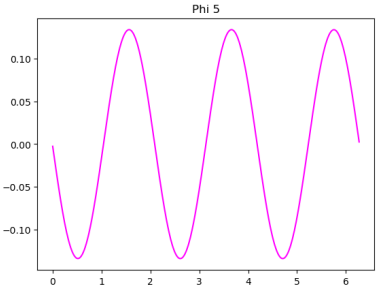
(b) ϕ_2



(c) ϕ_3



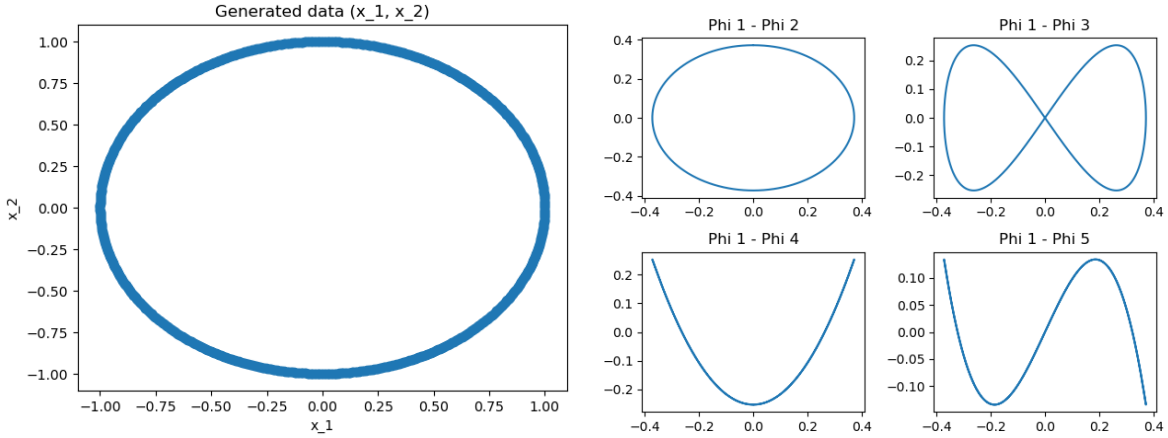
(d) ϕ_4



(e) ϕ_5

Figure 9: Separate Diffusion Map results

1.2: Bonus We see that the diffusion map can perfectly replicate the input data, as ϕ_1 and ϕ_2 represent the input data perfectly when disregarding the phase-shift. We can confirm this observation by plotting the input data as $(x_1, x_2$ and then comparing it to the diffusion map embedding (ϕ_1, ϕ_2). In figure 10 we observe that the manifold of the input data is perfectly captured by the diffusion map.



(a) Plot x_1 against x_2

(b) Plot ϕ_1 against ϕ_k, $k \in \{2, 3, 4, 5\}$

Figure 10: Different plots of Fourier data

2: The "Swiss Roll" data set In the second part we want to use the algorithm to obtain the first 10 eigenfunctions of the Laplace-Beltrami operator on the "swiss roll"-manifold. The data points u_k and v_k are sampled uniformly at random and define data set as follows:

$$
\begin{aligned}
X &= \{x_k \in \mathbb{R}^3\}_{k=1}^N \\
x_k &= (u_k \cos(u_k), v_k, u_k \sin(t_k)) \\
u_k &\in [3/2\pi, 3\pi] \\
v_k &\in [0, 21]
\end{aligned}
$$

In the following experiment all data points were sampled using the `sklearn` library function:

`sklearn.datatsets.make_swiss_roll(...)`.

The function argument `random_state=69` was set to have reproducible results. Also `sklearn` outputs an additional vector containing the univariate position of the sample according to the main dimension of the points in the manifold. This allows us to visualize the data points colored according to their univariate position. Hence we did not use our own implementation for creating sample points.

The experiment was performed with setting `max_dist=60.0`. Considering to few neighbors for the distance matrix leads to similar results as having little amounts of sampling data. Also note that due to points being sampled at random, the points are presented using a scatter plot.

2.1: 3D plot of "swiss roll" In figure 11 we see a 3D plot of the "swiss roll" data set with $N = 5000$ sampled points.

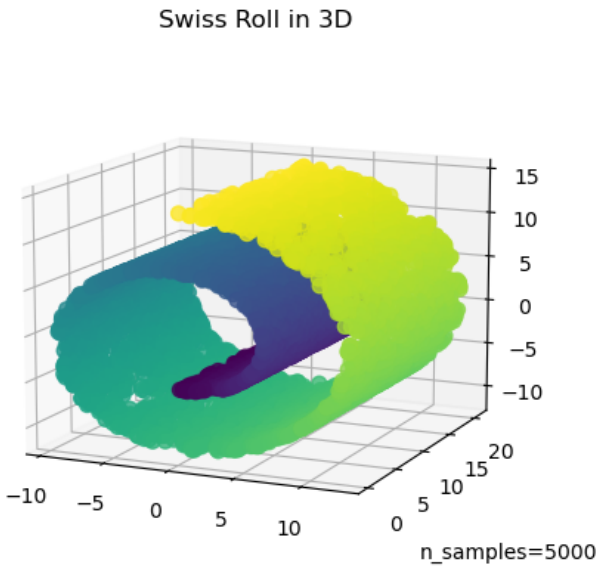


Figure 11: Swiss Roll with $N = 5000$ sample points

2.2: 10 eigenfunctions on swiss roll plotted on ϕ_1 Obviously ϕ_1 plotted against itself will show the linear function $f(x) = x$, hence we excluded this plot. In Figure 12 we see ϕ_1 plotted against ϕ_2 to ϕ_{10}. We see that for the first 3 subplots with $l \leq 4$ the results look like polynomial functions of degree 2, 3 and 4. Clearly "swiss-roll" is unrolled, but the sample points were mapped into a 1D space.

2.3: Number of l eigenfunctions where ϕ_l is no longer a function of ϕ_1? For $l = 5$ in the subplot "Phi 1 - Phi 5" we observe the "swiss-roll"-manifold unrolled as a plane in a 2d-space and ϕ_5 is no longer a function of ϕ_1. For $l > 4$ we see further attempts at unrolling appear in the next few scatter plots except for "Phi 1 - Phi 8". We observe that for higher l the diffusion map embedding has an linear increase in "knots".

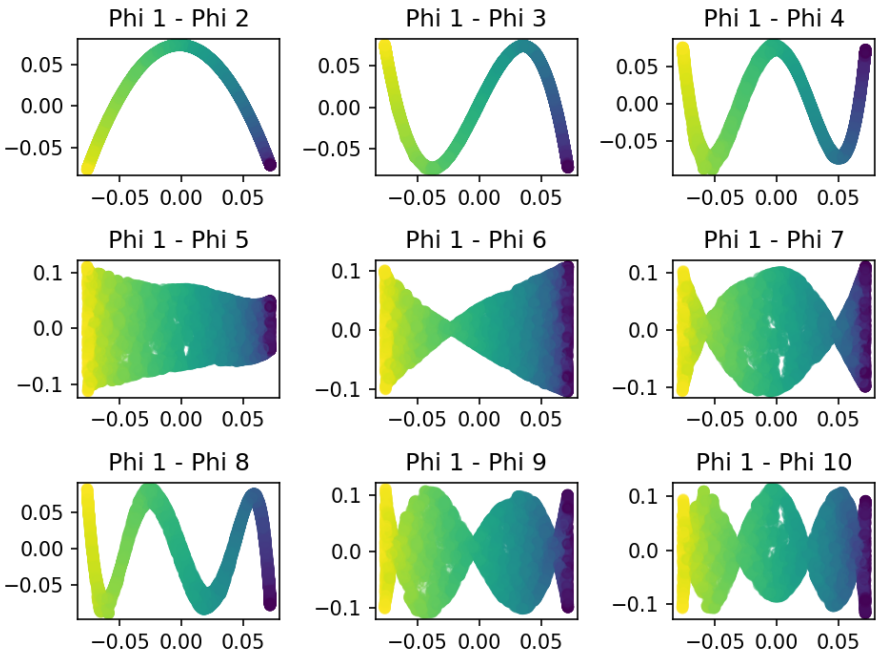
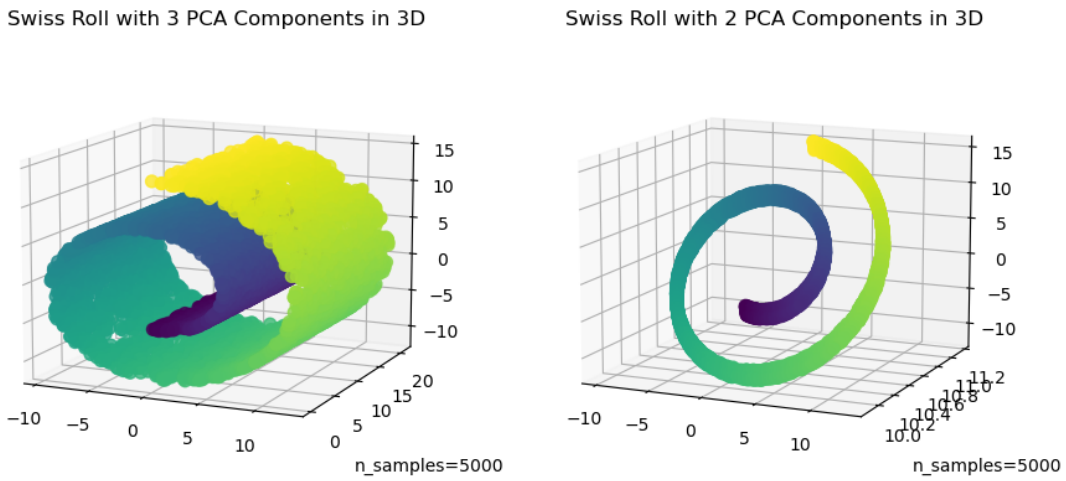


Figure 12: Scatter plot results

2.3: PCA on "swiss roll" with $N = 5000$ Next we used PCA with 3 and 2 components on the "swiss roll" data set for $N = 5000$ sample points. In Figure 13 we see that 3 components can perfectly capture "swiss roll", but 2 components are only able to capture a 2 dimensional projection of the "swiss roll".

2.4: Why do you need 3 principal components for the swissroll, not 2? The "swiss roll" is a 3-dimensional object and has variances in all 3 dimensions. Since PCA can neither discover nor make use of the 2 dimensional manifold of the "swiss roll", therefore, needs 3 components to reconstruct it.



(a) PCA with 3 components

(b) PCA with 2 components

Figure 13: PCA on "swiss roll" with $N = 5000$ sampled points

2.5: Diffusion Map on"Swiss Roll" with $N = 1000$ In this section, we repeat the previous experiment with fewer sample points $N = 1000$. In Figure 14 we see again a 3D plot of the "swiss roll" with $N = 1000$ sampled points. Comparing it to the "swiss roll" from Figure 11 we can clearly see some holes due to having significantly less sample points.

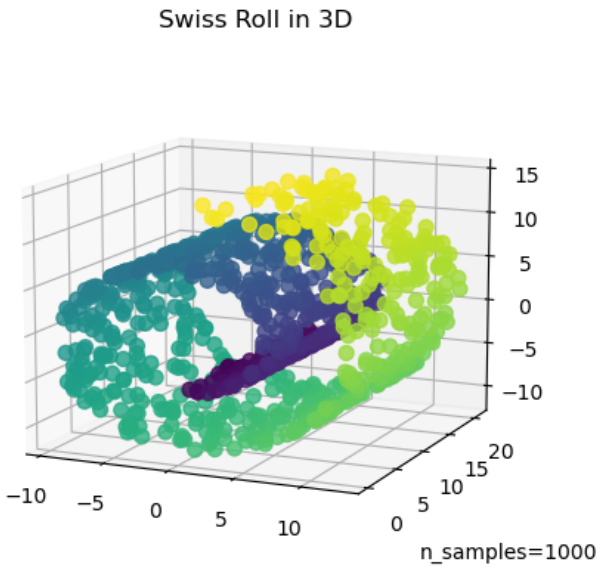


Figure 14: Swiss Roll with $N = 5000$ sample points

In Figure 15 we see ϕ_1 plotted against ϕ_2 to ϕ_{10}. Notieably with less sample points the diffusion map embedding degrades in quality. The previously observed functions now look poorly drawn. Also the unrolling we saw in "Phi 1 - Phi 5" can not be observed anymore. Similar results can be seen as well, when setting the `max_dist` for the distance matrix too low. Less data points can lead to different embeddings in the diffusion map. Similar to Fourier data set where we saw phase-shifts in the output, here we observe diffusion map points being mirrored, when comparing to 12.

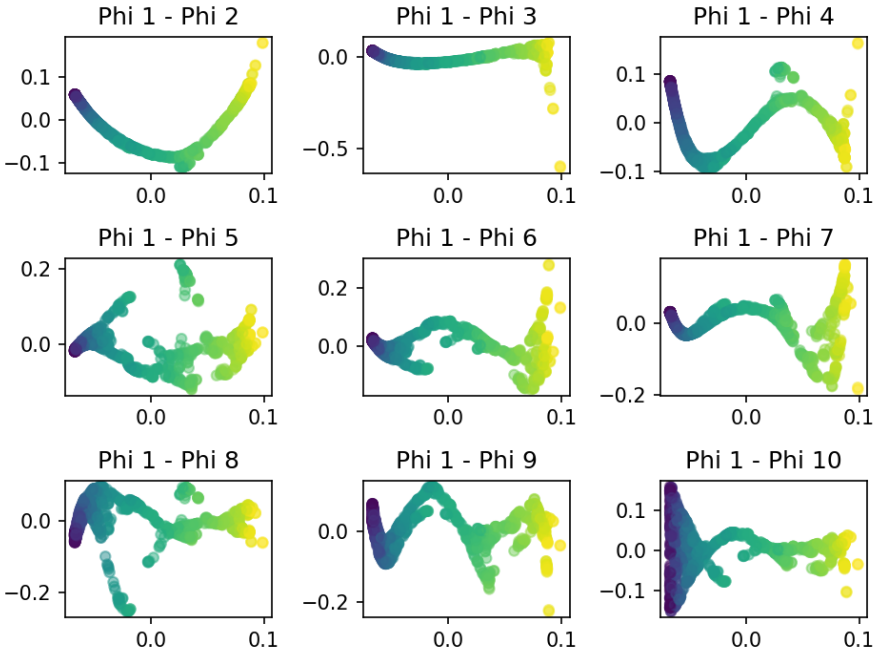
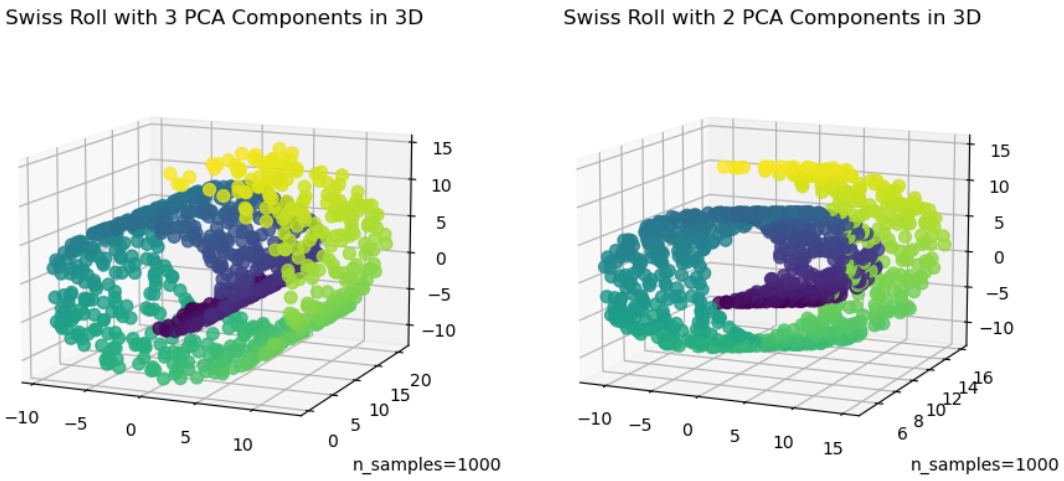


Figure 15: Scatter plot results

2.6: PCA with $N = 1000$ samples Here we repeat the same experiments on PCA with $N = 1000$ samples. In Figure 16 we can once again see a good reconstruction using 3 components. In Figure 16(b) the reconstruction with 2 components looks 3D, but we are actually observing high variance. With less samples, hence a lower confidence for the positions of the data points.

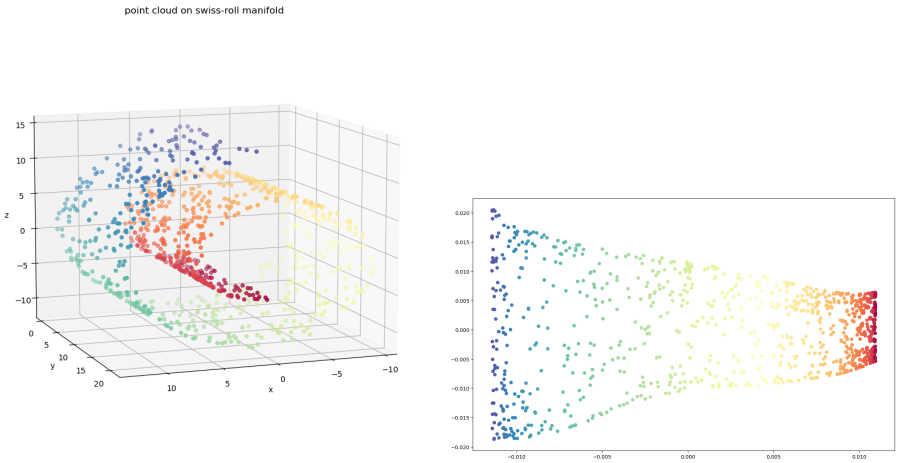


(a) PCA with 3 components

(b) PCA with 2 components

Figure 16: PCA on "swiss roll" with $N = 1000$ sampled points

2.7: Bonus - Diffusion Map on "swiss-roll" using `datafold` In this section we applied the diffusion map algorithm provided by the `datafold` library. We used the tutorial python notebook "Diffusion Maps: Embedding of a S-curve manifold" from `https://datafold-dev.gitlab.io/datafold/tutorial_03_dmap_scurve.html` as basis. The initial plot 17(a) of of the "swiss-roll" shows 1000 samples of the $N = 15000$ sampled points. This library is very efficient an manages to obtain the diffusion map embedding significantly faster than our implementation. Another nice feature of `datafold` is that it can automatically chose an embedding for you as seen in Figure 17(b).



(a) "swiss-roll" plot (b) Chosen embedding

Figure 17: Diffusion Map using datafold on "swiss roll" with $N = 15000$

In Figure 18 we see the first eigenfunctions plotted against other eigenfunctions. Comparing the results to ours from Figure 12 we see similar results from the diffusion map from the `datafold` library.

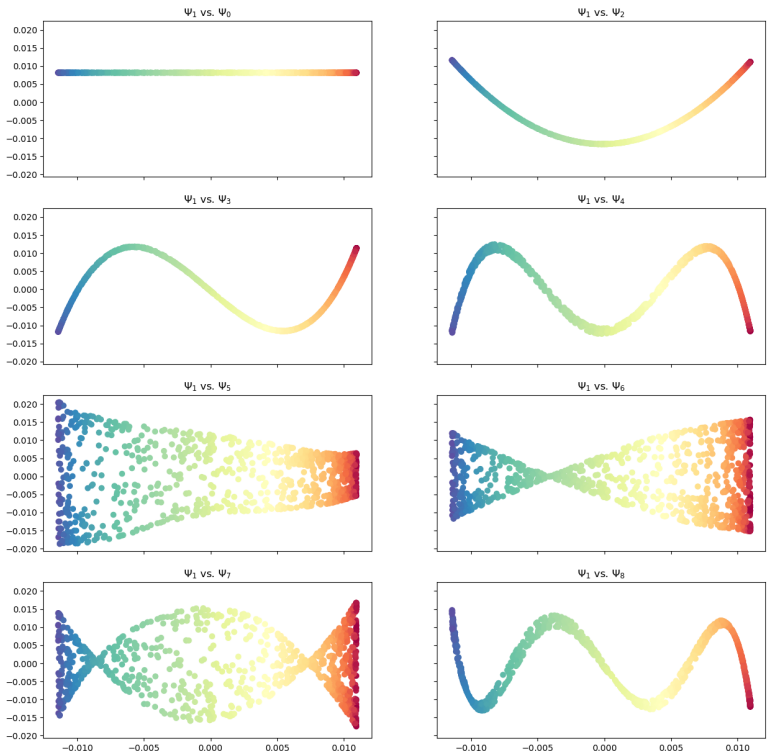


Figure 18: Plotting the eigenfunction against each other

3: The Vadere dataset In this part we want to perform the same analysis as we did in the previous section with PCA. Once again we have plotted all pedestrian trajectories in Figure 19. We can see that all pedestrians are walking around in a circle. This already tells us that the diffusion map embedding should be an ellipse.

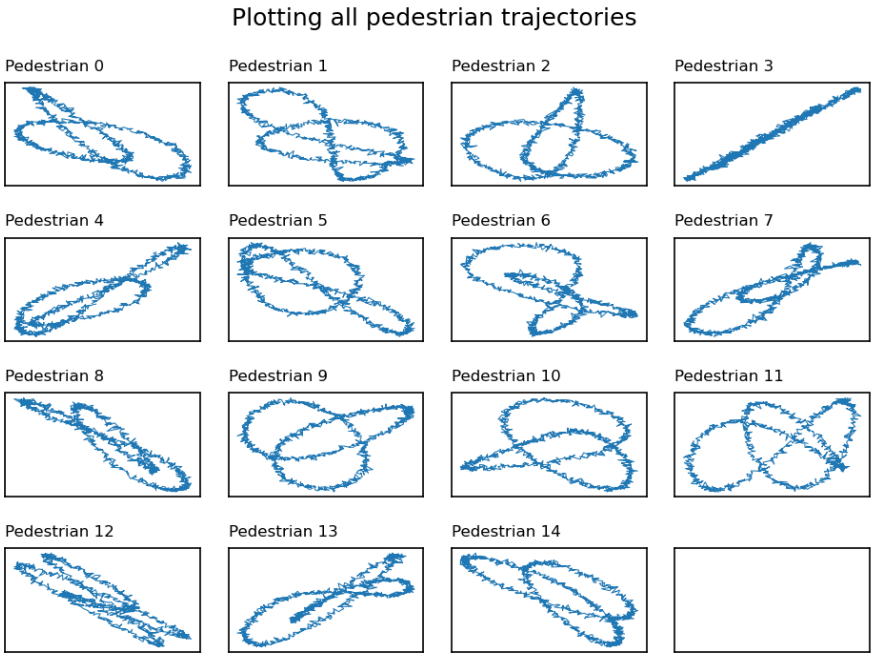
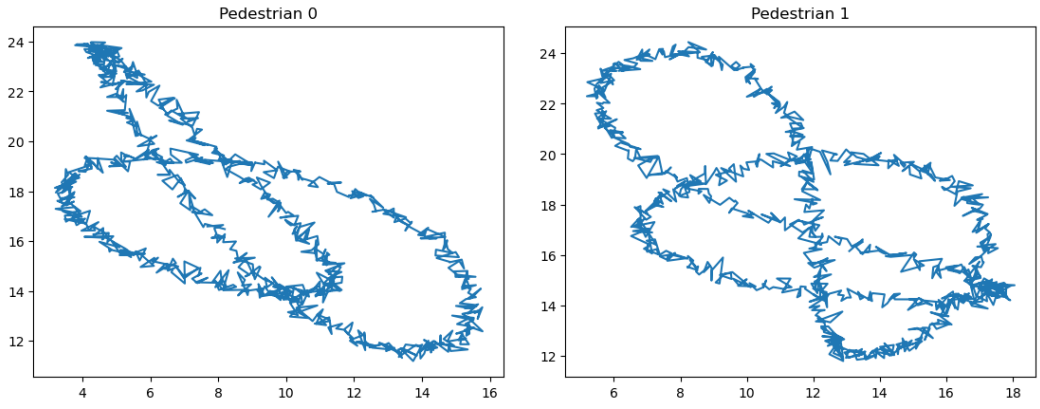


Figure 19: All pedestrian trajectories

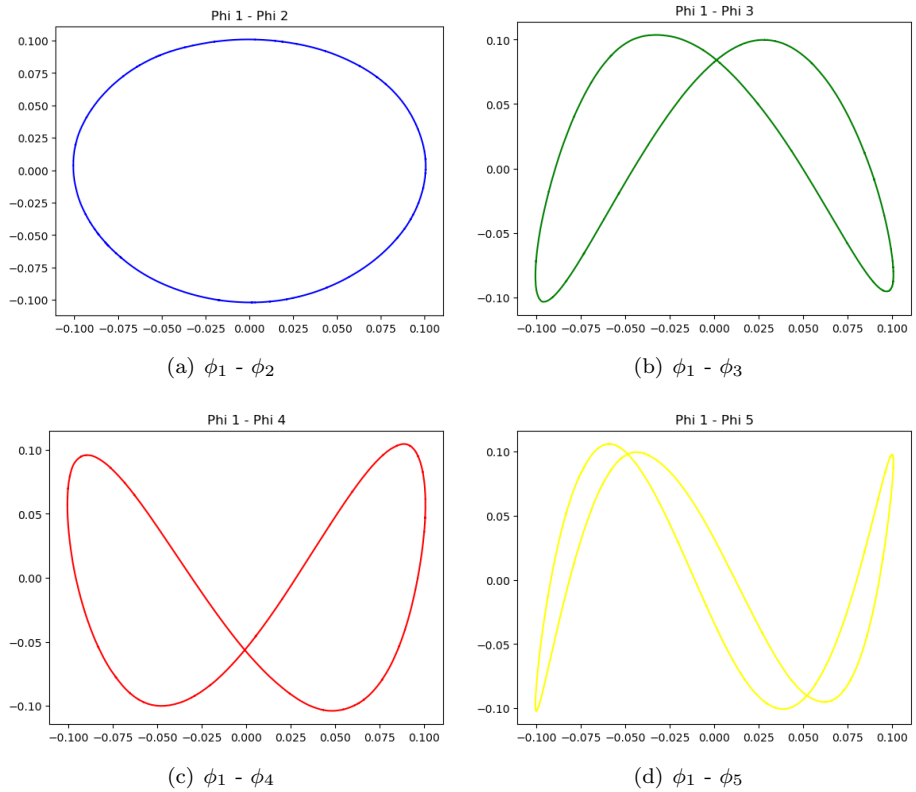
3.1: Visualize the path of the first two pedestrians in 2D space In Figure 20 we see the trajectories of the first 2 pedestrians from the `data_DMAP_PCA_vadere` data set.



(a) First pedestrian (b) Second pedestrian

Figure 20: 2 pedestrian trajectories

3.2: Are $l = 2$ eigenfunctions ϕ_l enough to capture the manifold of the data? As seen in Figure 19 or 20 we concluded that the pedestrian trajectories are circles. Therefore the diffusion map embedding should result in an ellipse. In Figure 21 we see that the first two eigenfunctions ϕ_1 and ϕ_2 capture the ellipse. Therefore 2 eigenfunctions ϕ_l are enough to capture the manifold contained in the pedestrian trajectory.



(a) ϕ_1 - ϕ_2 (b) ϕ_1 - ϕ_3

(c) ϕ_1 - ϕ_4 (d) ϕ_1 - ϕ_5

Figure 21: Diffusion Map results

3.3: Why, or why not? How many do you need to capture most data set? The first 2 eigenfunctions are enough to capture the pedestrian trajectory, because this trajectory 20 boils down to walking in a "circle". The diffusion map unfolds that trajectory into an ellipse, which in Figure 21(a) is roughly a circle when considering the values on coordinate axes.

Answering the three questions from the first page of the exercise sheet

- (a) an estimate on how long it took you to implement and test the method. About 12h.
- (b) how accurate you could represent data and what measure of accuracy you used. According to [1] we measure accuracy of diffusion maps by compare the diffusion distance against the euclidean distance of the diffusion map embedding. Formally:

$$D_t(x_i, x_j)^2 \approx ||\Phi_{\alpha,t}(x_i) - \Phi_{\alpha,t}(x_j)||_2^2$$
$$\Rightarrow |D_t(x_i, x_j) - ||\Phi_{\alpha,t}(x_i) - \Phi_{\alpha,t}(x_j)||_2^2| < \delta$$

  We follow [6] for estimating the accuracy using the mean squared error. Accuracy calculations can found in the python notebooks right after fitting the data. All results are within expectations:

Test	Fourier (ϕ_1, ϕ_2)	SR 5k (ϕ_1, ϕ_5)	SR 1k (ϕ_1, ϕ_5)	SRD 5k (ϕ_1, ϕ_5)	Vadere (ϕ_1, ϕ_2)
MSE	0.799	14.976	14.914	14.992	25.197

("SR" - swiss roll with sample size, "SRD" for swiss roll using datafold with sample size)

- (c) what you learned about both the data set and the method (which is probably different from what the machine learned). As seen from the visualization 19 the data seems to contain people walking roughly along some curve. Diffusion maps work extremely well in discovering in the underlying manifold, which is an ellipse (or maybe a circle).

Report on task 3, Training a Variational Autoencoder (VAE) on MNIST

Implementation For this task the model is implemented in `vae/model.py`. Also, to make the code more maintainable and modular, the model is trained using another file, `engine.py`, which takes care of running the train and test iterations and providing feedback to the user of the performance measures. In addition, we have support modules such as the visualisation module, the data processing module, or the tool module able to save the weights of the model in a folder. As a user, you should only interact with the `vae.ipynb` notebook to train and test the model.

Our implementation of the VAE is based on the publication [3]. It is composed mainly by an encoder, that approximates the posterior distribution, $q_\phi(z|x)$, and the decoder that tries to learn the distribution $p_\theta(x|z)$ in order to learn the latent representation and the underlying distribution of the input data by optimizing the parameters (ϕ, θ). Therefore, apart of trying to learn a correct encoding of the data, we will try to make the approximate posterior distribution to be similar to the assumed distribution of the prior $p(z)$ by minimising the KL divergence between the two distributions [4].

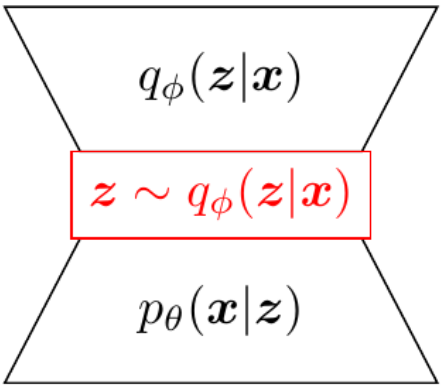


Figure 22: Variational Autoencoder structure

In order to do that, our Variational Encoder outputs the mean μ_e and standard deviation σ_e of the approximate posterior, while our Variational Decoder outputs the mean μ_d of $p_\theta(x|z)$, while the standard deviation σ_d is computed as a trainable parameter inside the model. Finally, the VAE model outputs the multivariate diagonal Gaussian distribution of $p_\theta(x|z)$, using μ_d as mean and the diagonalized σ_d^2 as covariance matrix, and the mean μ_e and the covariance matrix using σ_e^2 for $q_\phi(z|x)$ in order to be able to compute the KL divergence in the loss. z is sampled from the approximate posterior distribution using the pytorch function `rsample`, which is differentiable.

The chosen dataset to test the model is MNIST, a set of images of handwritten numbers. We will try to learn a latent representation of the data which we can reconstruct, while also trying to make the latent representation continuous enough to be able to generate new samples.

Figure 23: Sample of images of MNIST

Task 3.1 Regarding the implementation of the encoder and decoder, we needed to choose the activation functions to compute the mean and standard deviation of each distribution.

For μ_e we use a linear activation, as we don't want to impose any constraints on the values of the latent representation. This way allows the encoding to be an unconstrained real number. However, the standard deviation is constrained to be a positive number. Then we apply the exponential function e^x to obtain σ_e.

In the decoder we need to obtain μ_d and σ_d. Again, the mean is an unconstrained real number. However, we can change its activation function taking into account some external information of the dataset. We are

normalizing MNIST pixels to have values between [0,1] so we can constrain $\mu_d \in [0, 1]$ using the sigmoid function (Figure 24). Now, the most likely value for a pixel would be already constrained between 0 and 1. On the other hand, the standard deviation for the decoder follows the same rule than in the encoder, therefore we again apply the exponential function to make σ_d always positive.

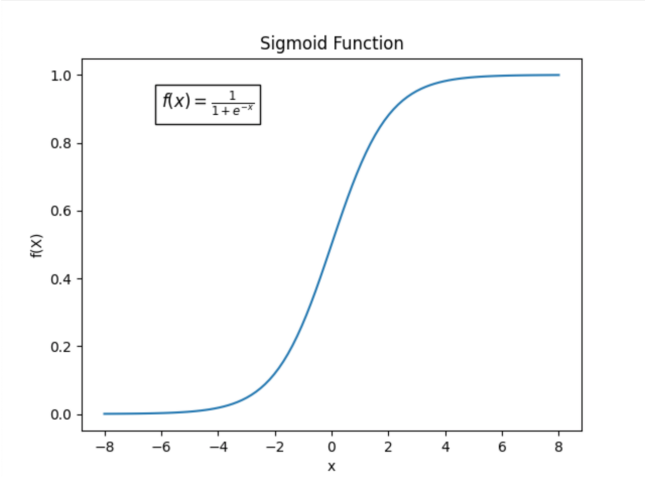


Figure 24: Sigmoid function

Task 3.2 We can deduce the cause of obtaining good reconstructed images but bad generated digits. First, we know that the objective function for the VAE is:

$$\mathcal{L}(x; \phi, \theta) = E_{z \sim q(z|x)} \log p(x|z) - \mathrm{KL}[q(z|x)||p(z)]$$

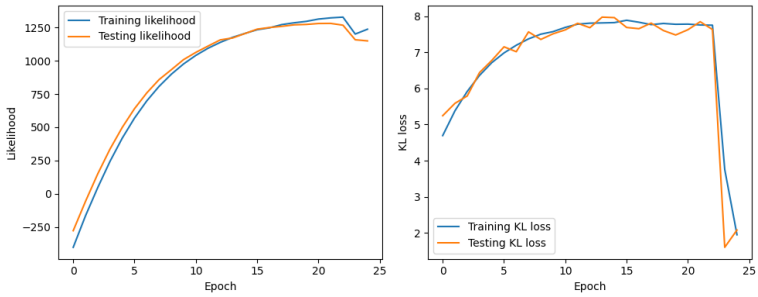
The first term controls how similar the distribution of x is given a z-sampled from the approximate $q(z|x)$. This term encourages the VAE to generate samples from the latent space that can accurately reconstruct the input data. The second term measures how similar are the distribution $q(z|x)$ from the objective prior distribution $p(z)$. When the second term is omitted, the VAE acts like an ordinary autoencoder, reconstructing the images from a latent vector z, without applying any constraint to the latent distribution.

Therefore, we know that the ordinary autoencoders are good at reconstructing and bad at generating due to the latent space distribution being discontinuous, i.e. not being closer to an objective continuous distribution.

Then, we can infer that if our model is bad at generating it is because our latent space is not being properly compressed, i.e. the KL divergence is very high. This means that the approximate posterior is far away from the prior. This can happen when our data is complex and we have too much information to be compressed in the latent space, generating a complex latent distribution. This makes the VAE learn how to reconstruct images but incapable of working with randomly generated samples of latent vectors.

In addition, we can have the opposite problem. Having the KL divergence very low could mean that the distribution $p(x|z)$ is very similar to the distribution $p(x)$. This can happen is $q(z|x)$ ignores x and returns a z with 0 mean and unit variance, not compressing information into z and therefore making the decoder to not use the latent vectors for reconstructing. This causes the decoder to reconstruct a blurred image, an average of all the images of the dataset. This is represented in Figure 25 as this phenomenon actually happened to us during training. When reaching 20 iterations in an old model the KL loss drastically dropped, making the model reconstruct images as the interpolation of all images of the dataset.

(a) Bad reconstructed digits at iteration 25



(b) Likelihood and KL loss

Figure 25: Effects of very low KL divergence

Task 3.3 Now, we will test our VAE. We will study how well it reconstructs and generates images at each epoch. We will also study the latent representation structure.

(a) MNIST digits

(b) Reconstruction at epoch 0

(c) Reconstruction at epoch 1

(d) Reconstruction at epoch 5

(e) Reconstruction at epoch 25

(f) Reconstruction at epoch 50

Figure 26: Reconstructed images with 2-dimensional latent space

In Figure 26 we can observe how the VAE is reconstructing the same sample of MNIST images. At first the images are a little blurry and some of them seem to be a mixture or interpolation between two or more digits.

When reaching 25 epochs all reconstructions are a single digit, without interpolations appearing. Nevertheless, some digits are "mistaken" from others, like the 5 that is reconstructed like an 8, or the two 4 that are reconstructed like 9. This could be related to the encoding of this two numbers being close to the mistaken numbers' encoding. We will explore this later.

(a) Generation at epoch 0

(b) Generation at epoch 1

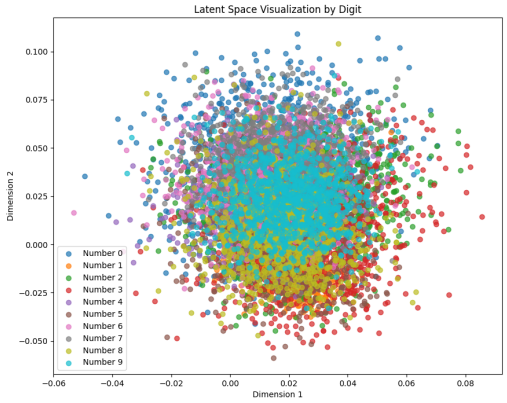
(c) Generation at epoch 5

(d) Generation at epoch 25

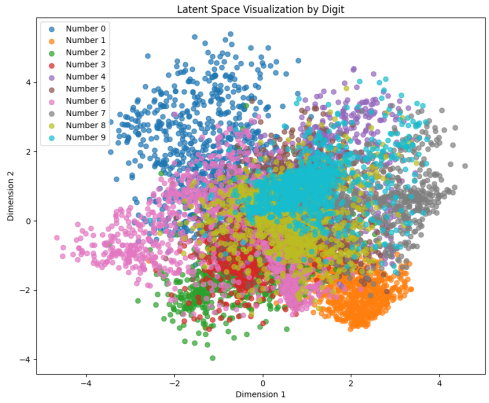
(e) Generation at epoch 50

Figure 27: Generated images with 2-dimensional latent space

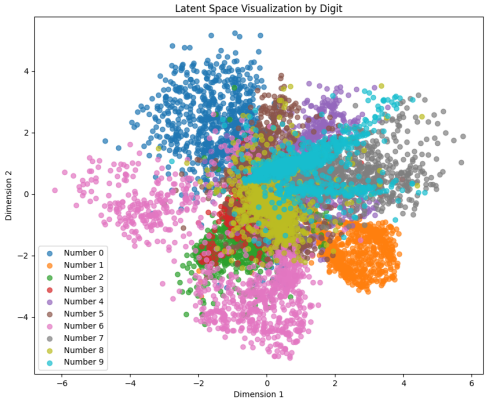
In Figure 27 we can check that our model is better at reconstructing than generating. The generated images are more blurred in the beginning, generating interpolations of digits. After some epochs the model generates clearer digits, however not all of them are perfect. we can see that the digit 6 appears repeatedly, as well as some digits that do not correspond to any real digit.



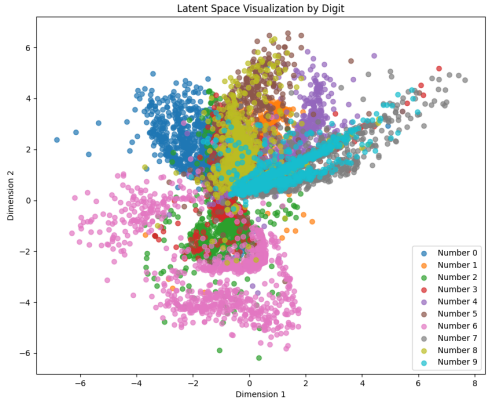
(a) Latent space at epoch 0



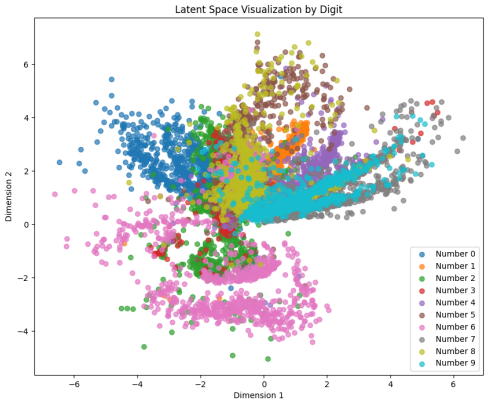
(b) Latent space at epoch 1



(c) Latent space at epoch 5



(d) Latent space at epoch 25



(e) Latent space at epoch 50

Figure 28: 2-dimensional latent space

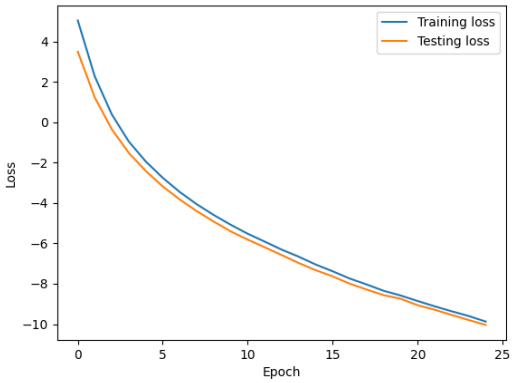
In Figure 28 we can observe the evolution of the latent space created by the VAE. At first all the encodings are distributed without an apparent pattern, but quickly thanks to training and the KL regulariser, the latent representations of the different digits start to separate, providing better reconstructions but above all better digit generations, the purpose of VAE.

Analysing the latent space, we can see how it reflects certain behaviours of our model. For example, the encoding of the number 6 is the most separated and distributed from the rest of the distribution, making the model less likely to be confused in this digit. In this way, we can see how the model properly reconstructs the number 6 in Figure 26 and how it generates it repeatedly in Figure 27e, since having the encoding quite spread out makes it more likely that this number will be generated. Another example of this is the number 0, which is usually reconstructed without problems, due to a good encoding that does not overlap with other digits.

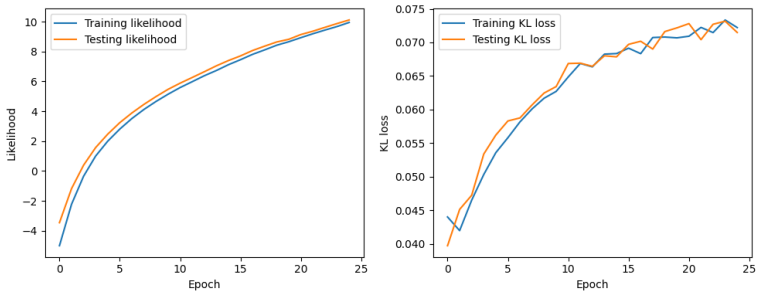
On the other hand, we can observe why our model makes mistakes with some digits. For example, it usually mistakes a 4 for a 9 in the reconstruction. If we look closely to Figure 28d or 28e, we will notice that the encodings for the digit 4 and 9 are mostly overlapping, making it less likely that the VAE will recover the correct image.

In summary, we can see how our latent space encodes a distribution that allows us to separate certain digits such as 6 and 0, but suffers with others due to a central area where many latent representations of different numbers overlap.

Task 3.4 Now, we will plot the loss or $-\mathcal{L}_{\mathcal{ELBO}}$. It is important to previously understand that minimizing the $-\mathcal{L}_{\mathcal{ELBO}}$ is equivalent to maximizing the likelihood of the output distribution for the sampled z and minimizing the KL divergence between the approximate posterior and the prior.



(a) Loss for 25 epochs



(b) Likelihood and KL loss

Figure 29: Training and testing loss for 25 epochs

In the Figure 29 we can observe how the loss is minimized throughout the training, while both the likelihood and the KL loss tend to increase. There are a couple of things that are interesting to note.

First, we can see how the testing loss is slightly smaller than the training loss, as well as the testing likelihood is slightly greater than the training one. After checking that the inner functioning of the model is correct, we can explain that this could happen because the training and the test data are very similar in MNIST, as it is composed of the same 10 numbers written in a lightly different way, but where the average distribution for each digit should remain. Therefore, the likelihood for training and testing are almost the same, but when this likelihood is subtracted to KL, which is on average higher in testing than in training due to a more random

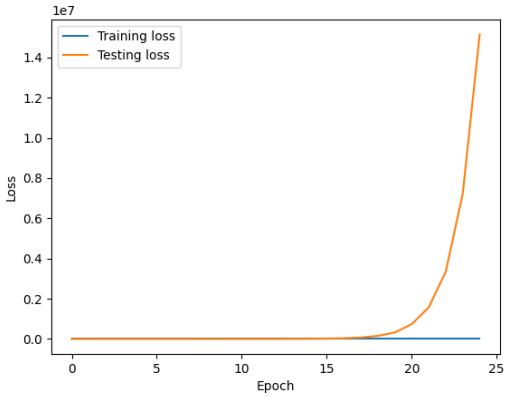
behaviour, produces this small counter-intuitive difference between both losses.

Second, we can observe how the KL loss is increasing instead of reducing, but in a very small magnitude, being almost constant. This can be happening because the actual value of KL is small and the latent encodings need to be less compressed in order to obtain better reconstructions, thus making smaller changes to the latent space sparsity. This can be noted in Figure 28 where the distribution starts with a very clear Gaussian structure and then gradually dissipates. Thus, it seems that the likelihood is the dominant term in the loss function.

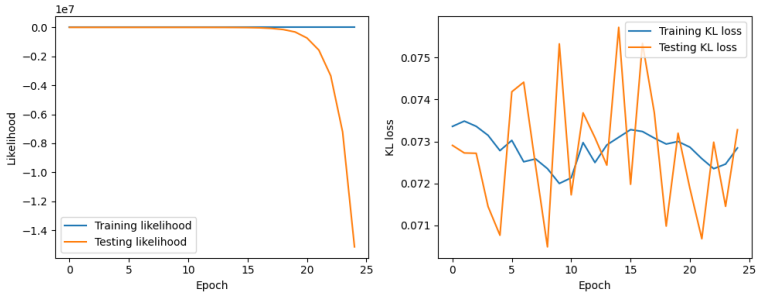
Finally, after training the model for 25 epochs more, we can observe how the testing likelihood drastically dropped, while the KL divergence remains almost constant (Figure 30, between 0.071 and 0.075. This could mean that the model is overfitting, being more specialized in the training dataset.

However, the reconstruction seems to be just as good as the one for 25 epochs. They are almost identical, showing that the model is not improving anymore. Nevertheless, the reconstruction is not bad in spite of having a drop in likelihood. Therefore, we think that this change may not be caused by overfitting but by numerical stability or exploding gradients in the model.

With more time available, a more in-depth study of the true cause of this strange behaviour could be done, but for the moment we have decided to assume that the model converges around 25 epochs, as the testing loss starts to increase after that.



(a) Loss for 25 more epochs



(b) Likelihood and KL loss

Figure 30: Training and testing loss for 25 more epochs

Task 3.5 Now, in order to explore the capabilities of the model with a bigger latent dimension, we have trained and tested the same model of the previous task but changing its latent dimension to 32.

This time, as we can observe in Figure 31 the reconstruction is almost perfect for the human eye, which sees the whole of the pixels. The reconstruction is far better than the one with a 2-dimensional latent space. We can infer that what causes such accurate reconstructions is a more expressive latent space, i.e. a latent space that allows for the representation of more complex information as it is not as compressed as two-dimensional space. Now, z can hold more relevant information to be able to reconstruct the images with more accuracy and detail.

Figure 31: Reconstructed images with 32-dimensional latent space

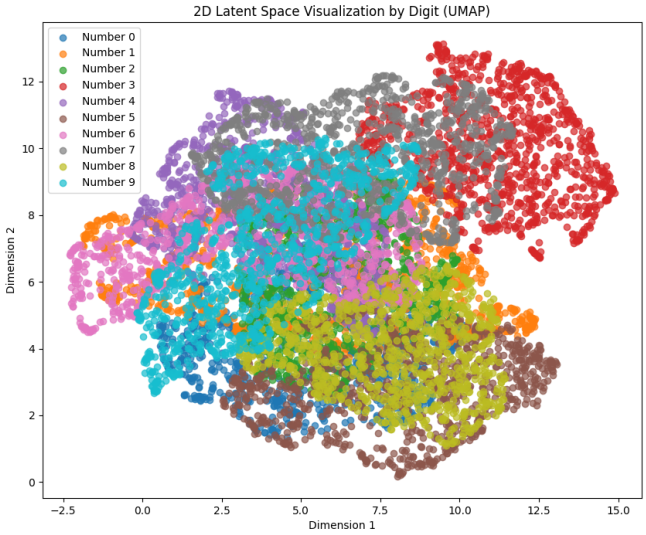
However, when generating images the model is worse than the one with 2-dimensional space as we can see in Figure 32. The reason for this may be the same as why we get good reconstructions. A higher dimensional latent space allows for better separation and identification of each number, but makes it more difficult for the model to transform a 32-dimensional random vector into a reasonable image, as there is a higher probability that the random vector is not close to any number distribution.

Figure 32: Generated images with 32-dimensional latent space

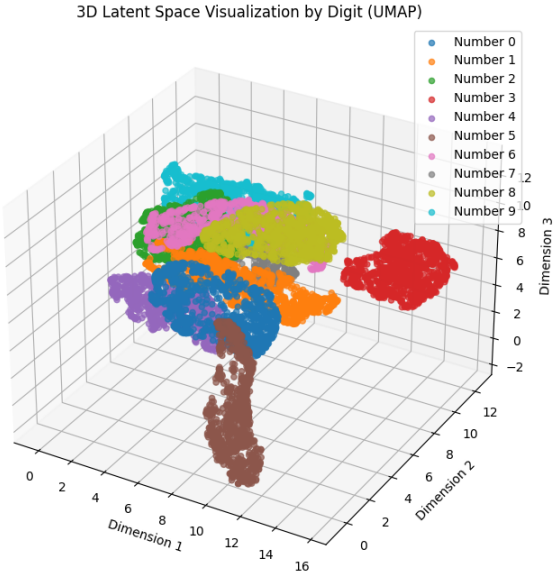
However, there is an interesting detail that can be noted. There are some numbers that are better generated than others, like the number 3. We wanted to study this phenomenon a little more so we decided to plot the latent space of the model. As it is a 32-dimensional space we needed to transform it into a displayable space, i.e. 2-dimensional or 3-dimensional.

We selected the UMAP algorithm as it aims to preserve both local and global relationships in which we are more interested [5]. Also it is based on manifold theory. Therefore UMAP was more appropriate than t-SNE for the analysis we wanted to do.

In Figure 33 we can observe the local and global relationships of the data in lower dimensions. In 33a we can observe how the number 3 distribution seems to be a little further away from the rest of the distributions. This fact is confirmed when the space is shown in 3 dimensions in 33b , which allows us to see how the number 3 is totally separated from the central distribution cloud. This allows that, when the random vector to be generated falls within the distribution of 3, it will be much easier for the model to identify it, generating a clearer picture without traces of interpolations with other numbers.



(a) 32-dimensional to 2-dimensional space with UMAP



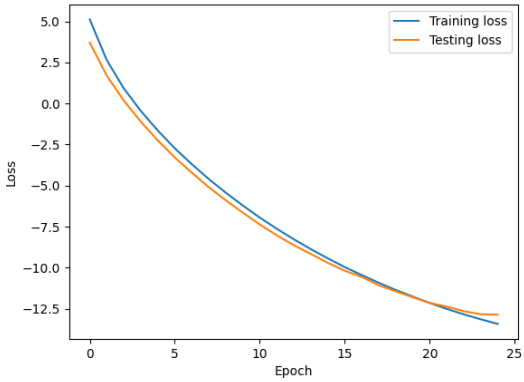
(b) 32-dimensional to 3-dimensional space with UMAP

Figure 33: 32-dimensional space projected into displayable dimensions

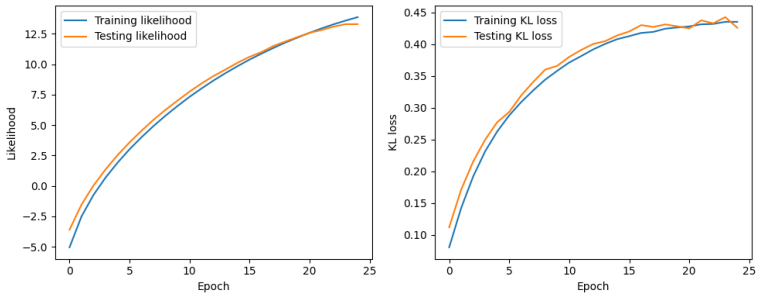
As we can observe in Figure 34 the loss evolves in a similar way than for the 2-dimensional model. The loss decreases each epoch, i.e. the likelihood increases at a bigger rate than the kl divergence increases. This indicates that the VAE architecture is easily scalable and that training does not suffer from increasing the dimension of the latent space.

Finally, after the 23 epoch, we can see how the training loss continues to fall while the testing loss remains similar, indicating that the model has reached its point of maximum generalisation. After 25 epochs, something similar to the previous model happened (Figure 30), the testing likelihood explodes and drops drastically, while the training likelihood continues to increase slightly. Therefore, we have used also the model with 25 epoch as

the converged model for the 32-dimensional space.



(a) Loss for 25 epochs



(b) Likelihood and KL loss

Figure 34: Training and testing loss for 25 epochs in 32-dimensional model

β-VAE implementation We had achieved a very good reconstruction with the 32-dimensional VAE, however we had not been able to generate digits at the same level. We wanted to try to improve our generation capacity and therefore, based on the "β-VAE: LEARNING BASIC VISUAL CONCEPTS WITH A CONSTRAINED VARIATIONAL FRAMEWORK" publication [2], we implemented a β-VAE.

Basically the β-VAE adds a parameter $\beta \in \mathbb{R}$ that controls the strength of the regularizer KL:

$$\mathcal{L}(x;\phi,\theta) = E_{z\sim q(z|x)} \log p(x|z) - \beta \mathrm{KL}[q(z|x)||p(z)]$$

Therefore, setting the β to 0 omits the KL divergence, thus converting the VAE in an ordinary autoencoder. Consequently, setting β to values below 1 decreases the importance of KL in the loss function making the latent space distribution less constrained. However, for values greater than one KL will have more regularising strength, resulting in a latent space with a distribution closer to the prior.

As we desired better generations, we wanted the latent space to be more constrained and closer to the prior distribution. Therefore, we did some test with $\beta > 1$. The most relevant are the following:

β testing for 2-dimensional VAE We tested different values of β as 10 and 30. We also tested smaller values but the results were very similar to $\beta = 1$, so we do not include them here.

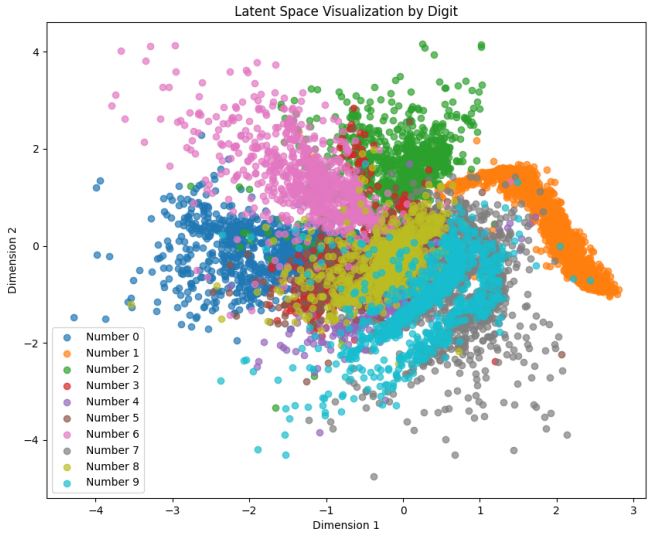
(a) Generated digits with $\beta = 1$

(b) Generated digits with $\beta = 10$

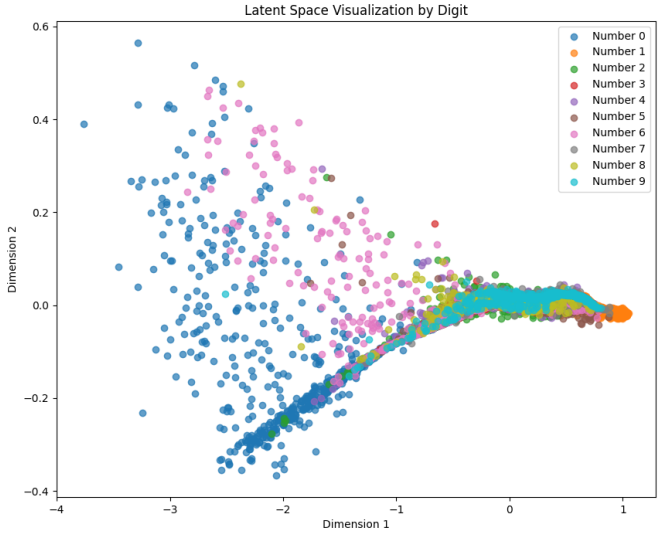
(c) Generated digits with $\beta = 30$

Figure 35: Comparing generated samples for different β values

As we can observe in Figure 35, after 15 epochs, the results using $\beta = 10$ have improved the results for generated digits, being them slightly clearer and defined. However, when increasing too much β we can observe how the KL is over regularized, constraining too much the distribution and making the model output an average image. We can further deepen the results by looking at the structure of the latent space.



(a) Latent space for $\beta = 10$



(b) Latent space for $\beta = 30$

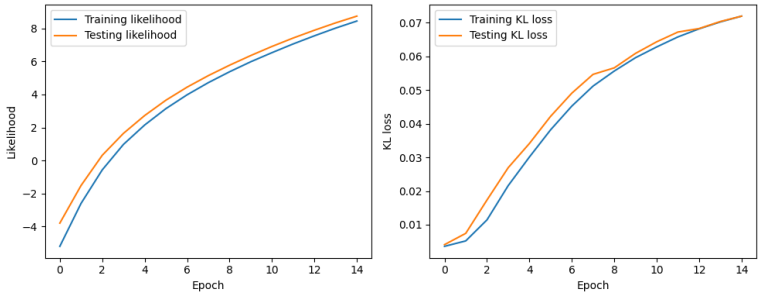
Figure 36: 2-dimensional latent spaces for different values of β

In Figure 36 we can clearly notice why the generated images of each model come out in their particular form. While the $\beta = 10$ model has a condensed but not too restricted latent space, the latent space of $\beta = 30$ is highly condensed in a small region, making the random vectors that fall outside it an average of the images in the dataset. On the other hand we can see how $\beta = 30$ generates well the number 0, which is just the distribution that spans most of the latent space.

This behaviour is in agreement with the explanation we gave in Task 3.2 when the KL is too low, where the model reconstructs average images of the dataset (Figure 37) where the KL values are very small. In addition, we can note for $\beta = 10$ how the reconstructions also slightly improved the ones of Task 3.2 using less epochs.

(a) Reconstructions for $\beta = 10$

(b) Reconstructions for $\beta = 30$



(c) Likelihood and KL loss for $\beta = 30$

Figure 37: Reconstructions for different values of β

β testing for 32-dimensional VAE The 32-dimensional VAE output the best reconstructions but the generated digits were very bad. Now, we try to generate better images for this model.

We tried again with $\beta = 10$ as it worked well in the 2-dimensional model.

(a) Generated images for $\beta = 1$

(b) Generated images for $\beta = 10$

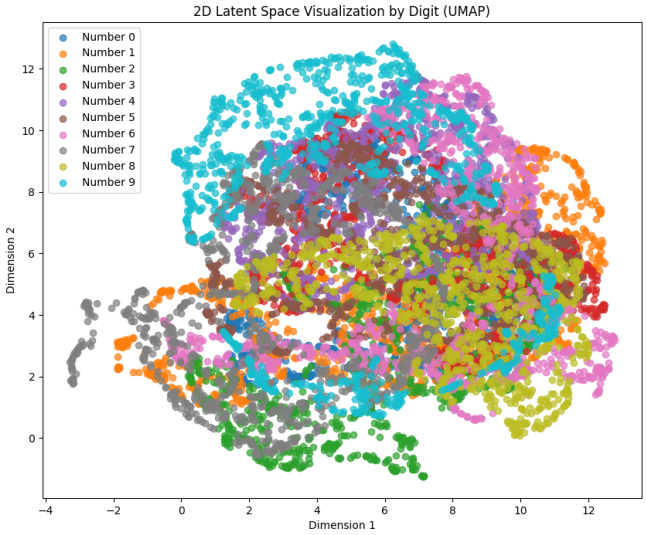
Figure 38: Comparison of generated images with the 32-dimensional model

As we can observe in Figure 38 the generated images are slightly better and clearer with $\beta = 10$. However, the reconstruction gets a little more and less detailed when constraining the latent space, as we can see in Figure 39. As we know, this happens because each number cloud is condensed together, making reconstruction more difficult but making it more likely that a random latent vector will have a detailed image associated with it (Figure 40).

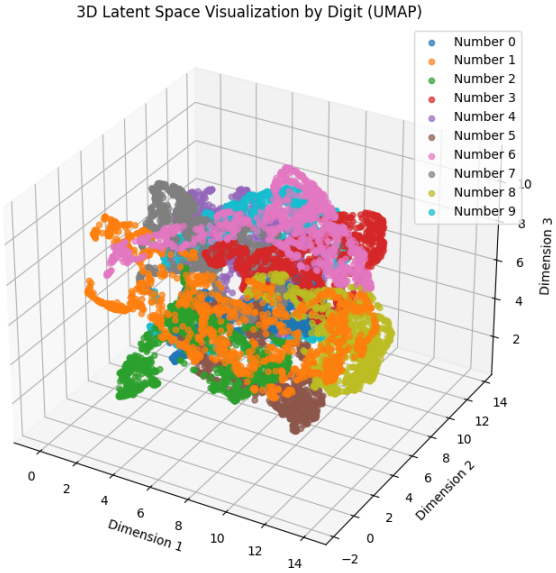
(a) Reconstructed images for $\beta = 1$

(b) Reconstructed images for $\beta = 10$

Figure 39: Comparison of reconstructed images with the 32-dimensional model



(a) 2d projection of latent space



(b) 3d projection of latent space

Figure 40: UMAP reduction of the 32-dimensional space for$\beta = 10$

Finally, we have explored the effects of adding the β parameter to the loss function that allows to apply more force to the KL regularization, producing a trade-off between reconstruction and generation making the latent space more or less similar to the prior distribution.

Answering the three questions from the first page of the exercise sheet

- (a) an estimate on how long it took you to implement and test the method. About 30h. The model last 1 minute per epoch so the accumulate time of testing is high.

- (b) how accurate you could represent data and what measure of accuracy you used. For the 2-dimensional VAE we got average reconstructions and generations. For the 32-dimensional VAE we got very detailed reconstructions but worse generated images. Finally we use the β parameter to get better generated images in the 2-dimensional and 32-dimensional model but at the expense of loosing reconstruction quality. As measures we used the likelihood of the reconstruction compared to the original image, as well as the Kl loss as an indicator of the latent space's structure. However, as we were working with images, we could use our own eyesight to discern whether reconstruction and generation were working well. Also, looking at the latent space representation we could infer if the model was working correctly.
- (c) what you learned about both the data set and the method (which is probably different from what the machine learned). We learned that the dataset was not as simple as it could seem. This can be shown when you need more than 2 latent dimensions to reconstruct the detailed image, because all the information in the image could not be reduced to 2 dimensions without losing part of it. Also, it was interesting to note the importance of the latent space structure in the performance of reconstructions and generations. Using the β parameter we could experiment with the trade-off between the performance on both quantities by manipulating the latent space structure, allowing us to understand better which structure benefits reconstructions and which one benefits generations.

Report on task 4, Fire Evacuation Planning for the MI Building

In response to the increasing student population at the Technical University of Munich, there is a pressing need to reassess the fire evacuation plan for the MI building (Figure 41). To address this, an experiment was conducted where the fire department tracked 100 random students and employees on different days during the busiest hours to obtain data on the distribution of people within the MI building, denoted as $p(x)$. The primary objective of the initial experiment was to estimate the number of people in a critical area of the building, marked by the orange rectangle in front of the main entrance, ensuring that it does not exceed 100 people. The experiment is divided into several steps, including data visualization, training a Variational Autoencoder (VAE) on the FireEvac dataset, visualizing the reconstructed test set, generating samples, and estimating the critical number of people. This series of experiments aims to gain a deep understanding of the distribution of people within the MI building through machine learning and simulation, providing robust support for the optimization of the fire evacuation plan.

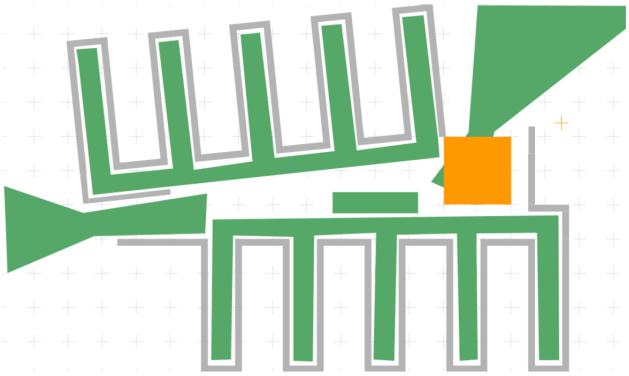


Figure 41: Schematic of the MI building in Garching

Visualise FireEvac Dataset To effectively visualize the data points in the FireEvac Dataset, we have plotted them as scatter points in Figure 42. There are 3,000 training data points, represented in blue, and 600 testing data points, represented in red.

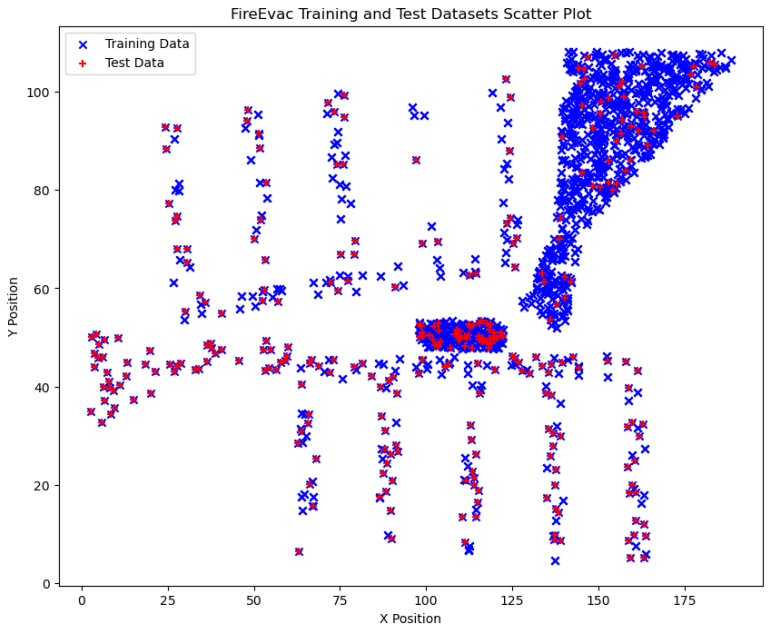


Figure 42: FireEvac Training and Test Dataset Scatter Plot

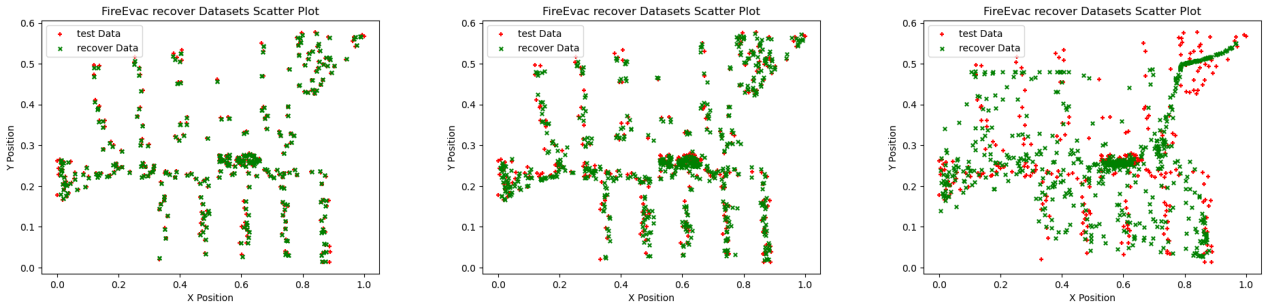
VAE traning on FireEvac Dataset In order to learn the probability distribution $p(x)$ on the FireEvac dataset, a Variational Autoencoder (VAE) was trained. The VAE implementation used in this task was adapted from Task 3, with adjustments made to the number of input and output neurons, and varying the number of layers. However, after some hyperparameter tuning, we changed the model slightly. In particular, we changed the activation function of the hidden layers from ReLU to LeakyReLU(0.2). Using the ReLU activation function the reconstructed results were very bad due to abruptly transforming negative inputs to 0. With the LeakyReLU we got a smooth transition when getting negative inputs and weights. The new model can be found in `model_fire_evacuation.py` with the only change being the replacement of the activation function.

To enhance model performance and take advantage of the sigmoid function in the decoder, the input data x was rescaled to a range of $[0, 1]$ before initiating the training process. For the architecture of the VAE, a configuration of 2-64-64-2 neurons for the encoder and decoder was employed. This translates to a two-dimensional latent space and two hidden layers, each comprising 64 neurons. The chosen learning rate was 0.005, utilizing the Adam optimizer with a batch size of 64. The model underwent training for 200 epochs.

Due to our loss being composed of two components, likelihood and KL-loss, where the KL-loss incorporates a scaling factor β, the loss is defined as loss $= -\text{likelihood} + \beta \times \text{kl\_loss}$. Because the performance of the trained model varies under different β parameters, we trained models with β values of 0, 0.5, and 1 to perform the following tasks.

Reconstructed Test Set The performance of the model in reconstructing the test dataset under three different β values is illustrated in Figure 43. When β is set to 0, the VAE degrades into an AE, and it almost perfectly reconstructs the test dataset, as depicted in Figure 43(a). This happens because the input dimension and the latent space dimension is the same, 2, so the latent space basically copies the distribution of the points in the original data, learning the trivial encoding. For $\beta = 0.5$, although there is a slight deviation between the reconstructed data and the original data, it essentially captures the pattern of the test dataset, as shown in Figure 43(b), as the regularizer KL is weak. When β is set to 1 43(c), the distribution of the reconstructed data is more dispersed compared to the original data, as the encodings are forced to be closer, breaking the trivial encoding. Therefore, the VAE is forced to learn a 2-d encoding closer to the prior distribution, of 2-d data points, a process that is quite inefficient. It is noteworthy that in this case, in the top right corner, the distribution of the reconstructed data forms a line, which is not ideal.

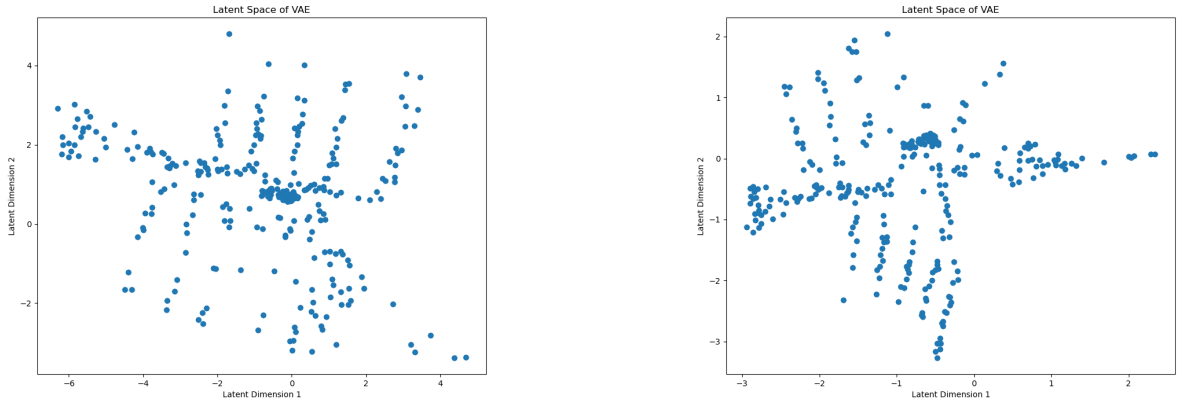
We can observe the differences in the latent space in Figure 44. We can note with $\beta = 0$ how the model tries to learn a trivial encoding imitating the distribution of the 2-d datapoints, whereas with $\beta = 1$ the latent space is constrained by KL so it learns an unprecise trivial representation of the datapoints, without finding an efficient encoding that is actually able to compress the information and reconstruct it. This is mainly because the input data already has a dimension of 2.



(a) $\beta = 0$ (b) $\beta = 0.5$ (c) $\beta = 1$

Figure 43: Reconstructed Test Set for Different β Values

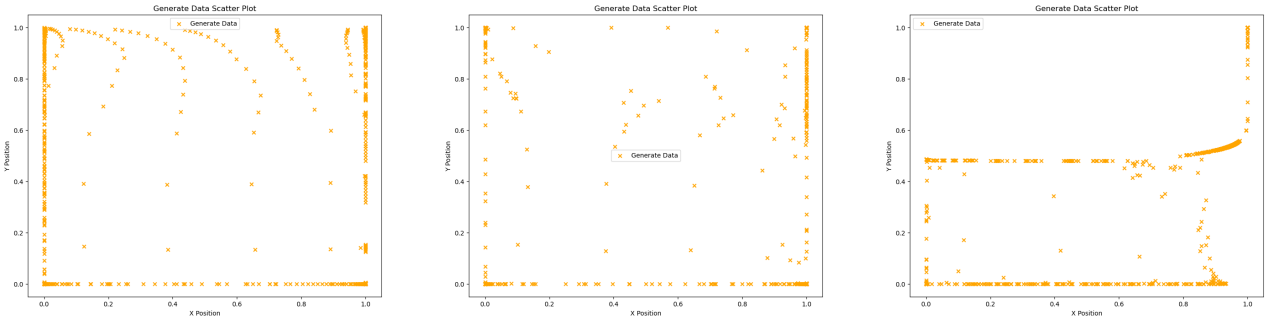
1000 generated samples Next, we plan to use the decoders from the models trained with three different β values to generate data points. To determine the numerical range to input into the encoder, we aim to understand the range of values when mapping the test dataset to the latent space. Thus, we input the test dataset into the encoder and visualize the two-dimensional latent space as a scatter plot, as depicted in Figure 44(b). From the graph, it is evident that when the test dataset is mapped to the latent space, the horizontal and vertical coordinates fall within the ranges of (-3, 3) and (-3, 3), respectively.



(a) Latent space $\beta = 0$ (b) Latent space $\beta = 1$

Figure 44: Latent space forβ Values

Therefore, we plan to use values within this range as input to the decoder to generate 1000 data points. We employed two methods to sample the latent space: normal distribution and uniform distribution. In the case of the uniform distribution, we uniformly sampled 32 points along the x-axis in the range $[-3, 3]$, and also uniformly sampled 32 points along the y-axis in the range $[-3, 3]$. This allowed us to generate $32 \times 32 = 1024$ samples using these two dimensions. Taking 1000 samples from this set and inputting them into the decoder resulted in the generation of 1000 synthetic data points. The generated data points are depicted in Figure 45.



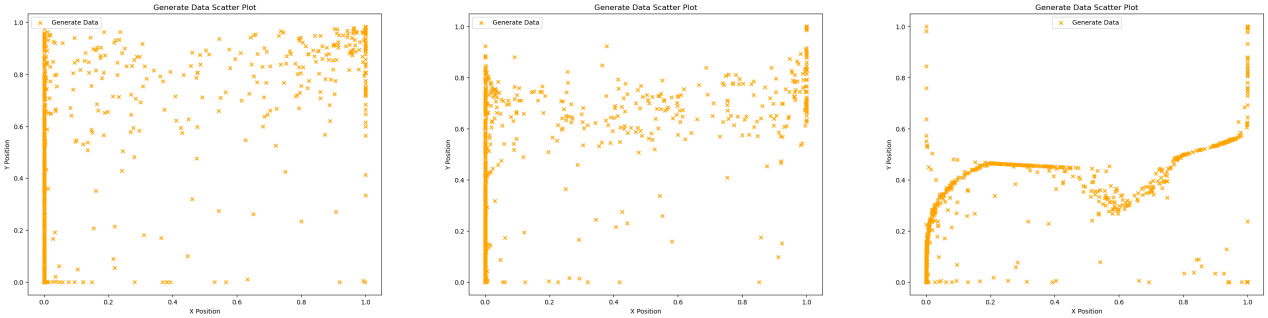
(a) $\beta = 0$ (b) $\beta = 0.5$ (c) $\beta = 1$

Figure 45: Generated Data for Model with Different β Values

Based on the previously demonstrated distribution of the latent space **??**, we can also consider it as a normal distribution. Therefore, we adopted a Gaussian sampling approach with a mean of (-0.5, 0) and a covariance matrix of $\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ to sample 1000 points. Subsequently, these samples are used to generate data, and the final generated results are depicted in Figure 46.

The models can somewhat reconstruct the test data. But from the generated data shown above, we observe that the models' ability to generate data is quite poor. Because we can hardly find patterns among them that resemble those present in the original data points. We speculate that the reason for this might be the insufficient quantity of training data or the model's inability to capture this particular pattern

Critical Number Estimation Since the previous models are unable to generate patterned data similar to that found in the FireEvac dataset, a team member of us attempted to build a new VAE model independent of Task 3, and making numerous attempts. Various experiments were conducted, including trying different β values, hidden layer numbers, latent space dimensions, hidden layer neuron numbers, and activation functions. However, none of the models trained from these attempts are able to generate reliable data. Using such data

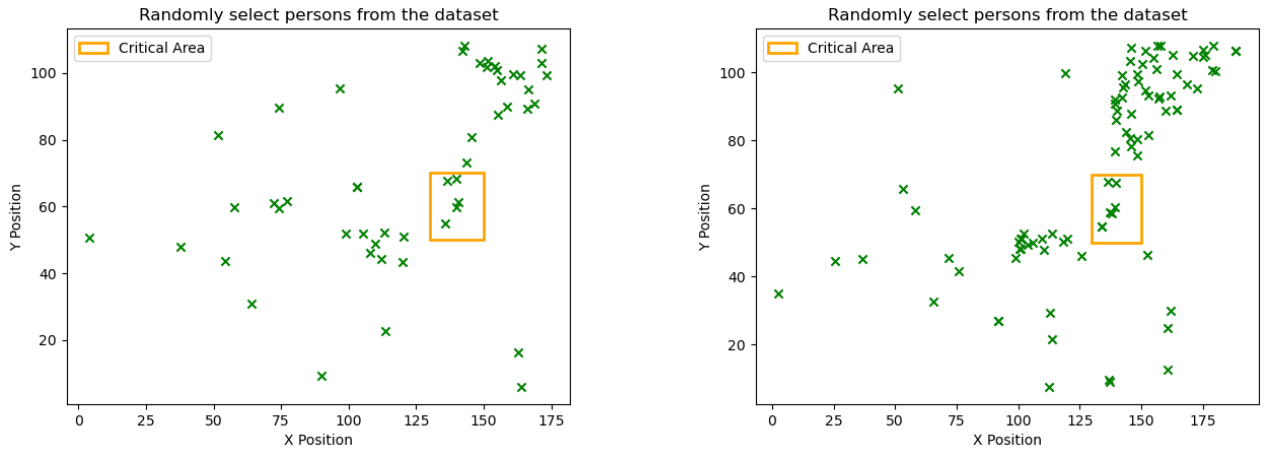


(a) $\beta = 0$ (b) $\beta = 0.5$ (c) $\beta = 1$

Figure 46: Generated Data for Model with Different β Values

for evaluation would be irresponsible concerning the safety of people inside buildings. Therefore, we prefer not to use data generated by this method for evaluating the critical number at the main entrance. Instead, we will rely on the dataset to assess the critical number at the main entrance.

Our strategy is as follows: We will start by merging the training and test sets of FireEvac. In our experiments, we generate one person at a time, and their location data is randomly selected uniformly from the merged dataset. The diagrams in Figure 47 illustrate the scenarios when 50 people are generated and when a count of 90 pedestrians has been reached, respectively.



(a) 50 people generated (b) 90 people generated

Figure 47: Randomly select persons from the dataset

If the generated pedestrian is located at the sensitive area in front of the main entrances, we increment the counter. When the count accumulates to 100, which is the Critical Number, we record how many people have been generated in total. Figure 48 illustrates the scenario when the critical area reaches 100 people, at which point a total of 1032 people have been generated.

To accurately estimate the number of peoples in the sensitive area in front of the main entrance that typically exceeds the critical number under such a distribution, we repeated the above experiment one hundred thousand times. We then calculated the average and median of these one hundred thousand runs. Their respective values are 1249.6 and 1246.0. In other words, when the number of peoples inside the building approaches about 1246, it reaches the critical number for the MI building.

Questions from the first page of the exercise sheet Since the VAE has already been implemented in Task 3, we are not including the time spent here. However, it is evident that the generated data is not ideal. Both of our team members collectively spent approximately 35 hours testing and refining the model in an attempt to

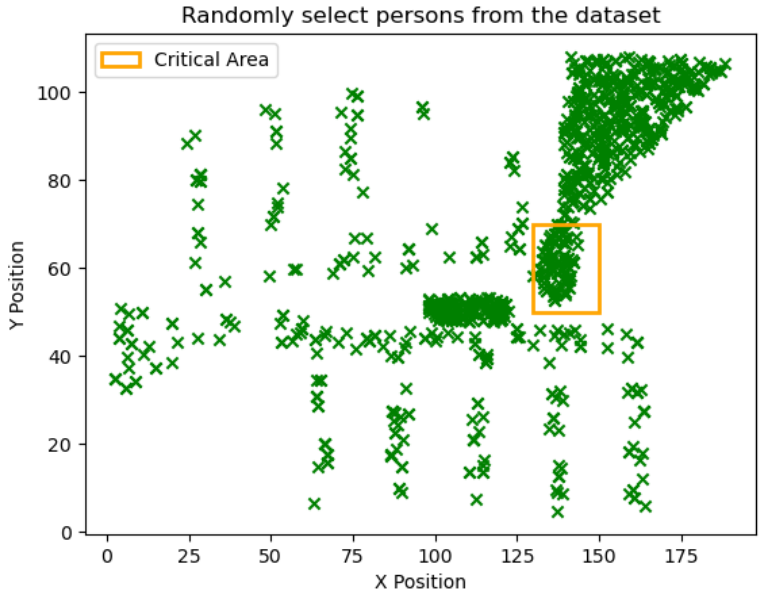


Figure 48: Critical area with 100 People

generate satisfactory data. Regarding the final critical number estimation, it took one hour to implement and test.

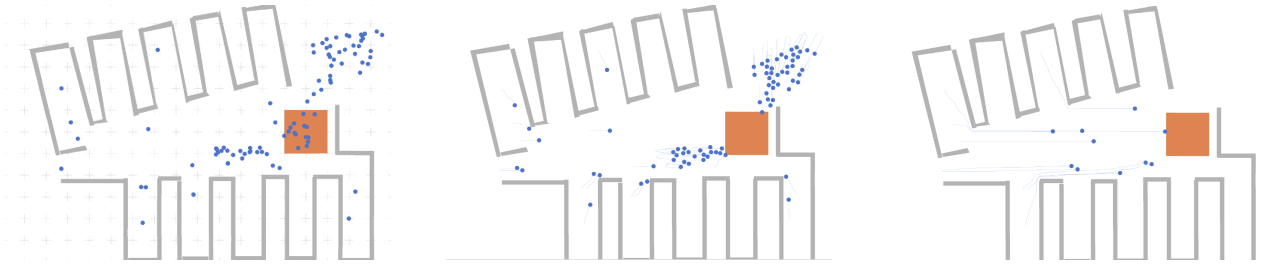
We chose not to explicitly incorporate a precision metric since the efficacy of outcomes is readily discernible through visual inspection of images. Additionally, given that the data comprises two-dimensional coordinates, its accurate representation on a flat plane inherently yields a high level of precision.

In this task, we learned how to train VAEs more effectively to enhance their performance. Additionally, we recognized that while VAEs are an excellent method for generating data, their performance may suffer when there is a shortage of training data.

Vadere scenario for the MI building By observing the scattered distribution of points in the FireEvac dataset, we created a map with dimensions of 120×180. Using the *source* tool in Vedere, we meticulously created a rectangle as an exit according to given coordinates. To easily identify the location of walls, we utilized the `add_pedestrian.py` tool created in Exercise 02 to add 1200 pedestrians to the scenario, adhering to the distribution $p(x)$. This facilitated the recognition of wall positions.

After creating the walls using the *Obstacle* tool, the previously generated 1200 pedestrians were removed. Subsequently, we used `add_pedestrian.py` again to reintroduce 100 pedestrians into the scenario, following the distribution $p(x)$. Thus, the scenario was successfully created. The visualization can be seen in Figure 49. The process of adding pedestrians has been documented in the `add_pedestrian_with_distribution.ipynb` file.

After successfully setting up the scenario, we tried running it through the GUI. However, we notice that each time we ran the scenario, the GUI display turned gray, which initially led us to suspect it might be due to an excessive number of pedestrians. Yet, after a wait, the movement of pedestrians became visible. Once the paths were completed, several files were generated and stored in the designated folder. We move the folder into the project directory for further examination. The paths of pedestrians can be observed in the graph 49. The paths taken by pedestrians do not surprise us and are consistent with the behaviour of the people we see daily on the TUM campus.



(a) start (b) progress 1 (c) progress 2

Figure 49: MI building scenario in Vadere

References

[1] Berry et. al. Time-scale separation from diffusion mapped delay coordinates. *SIAM J. Applied Dynamical Systems*, 12(2):618–649, 2013.

[2] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2016.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

[5] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[6] E. S. Roozemond. Dimensional reduction using diffusion maps. *U.U.D.M. Project Report*, 2021.