Report for exercise 3 from group H

Tasks addressed:	4
Authors:	Nayeon Ahn () Muhammed Yusuf Mermer () Milena Schwarz () Antonia Gobillard () Hammad Basit ()
Last compiled:	2024–05–18
Source code:	https://github.com/Hammad-7/MLCMS_Exercises.git

The work on tasks was divided in the following way:

Nayeon Ahn () **Project lead**	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
Muhammed Yusuf Mermer ()	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
Milena Schwarz ()	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
Antonia Gobillard ()	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
Hammad Basit ()	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%

Report on task 1, Principal component analysis

For task 1, we focus on Principal Component Analysis (PCA), which is a dimensionality reduction technique used in statistical data analysis and machine learning. It simplifies complex datasets with high dimensions while attempting to preserve the essential patterns and relationships. PCA focuses on identifying the principal components that capture the most variance within a dataset. These components are orthogonal vectors representing the directions of maximum variance. PCA involves calculating the covariance matrix of the data and its eigendecomposition. The eigenvectors of the covariance matrix correspond to the principal components, and the eigenvalues indicate the amount of variance captured by each component.

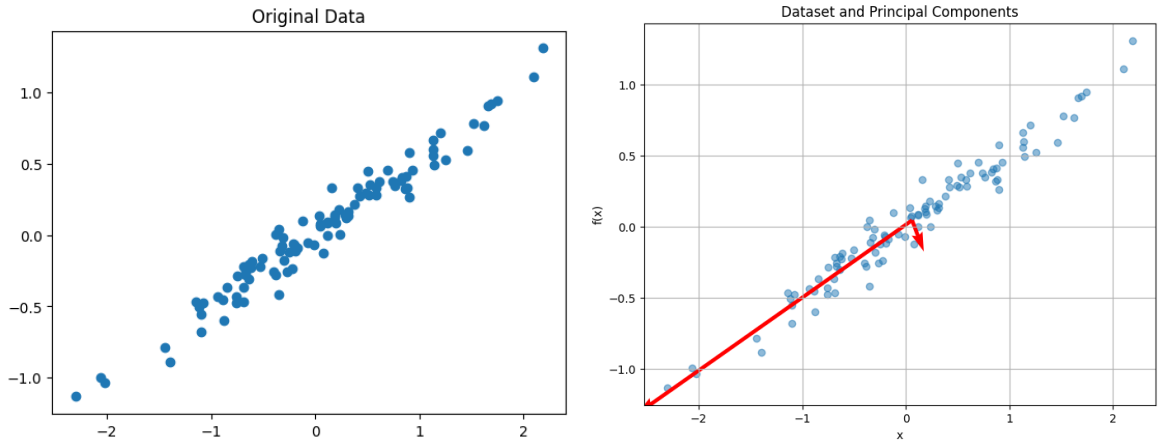
Given a dataset $\mathbf{X}$ with m samples and n features, PCA consists of the following steps:

1. Standardize the dataset.
2. Compute the covariance matrix $\Sigma = \frac{1}{m-1}\mathbf{X}^T\mathbf{X}$.
3. Find the eigenvectors ($\mathbf{v}$) and eigenvalues (λ) of Σ.
4. Sort eigenvectors by decreasing eigenvalues and form a feature vector $\mathbf{W}$.
5. Project $\mathbf{X}$ onto the new subspace: $\mathbf{Y} = \mathbf{XW}$.

But, PCA also has its limitations. It is sensitive to the scale of the features, emphasizing the importance of standardization. Moreover, PCA assumes linear relationships and may not perform optimally with non-linear data. Also, determining the number of components to retain is crucial, often based on the explained variance.

We are implementing PCA on three different datasets using our custom Python script, 'PCA.py', designed to perform Singular Value Decomposition. This file handles preprocessing needs, including data normalization, and also offers customization for selecting the number of components based on the dataset's characteristics. It evaluates PCA performance through metrics like explained variance, complemented by visual tools for result interpretation. Compatible with recent Python versions and dependent on standard libraries, it integrates into different analytical workflows.

- **Part 1:**
  Part 1 involves a two-dimensional dataset, `pca_dataset.txt`. We implemented PCA with 2 components. The variance explained by each component is [0.99314266, 0.00685734], summing to an overall variance of approximately 1.0.



(a) Dataset Plotted (b) Dataset and Principal Components Plotted

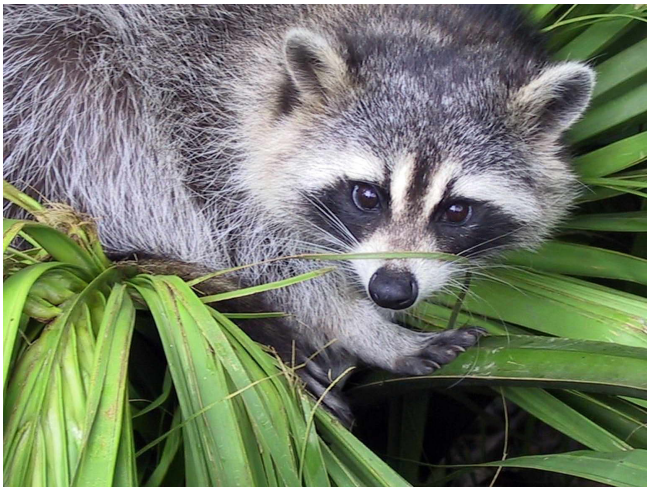
Figure 1: PCA on 2D Dataset

What we learned from the results of PCA (see figure 1).

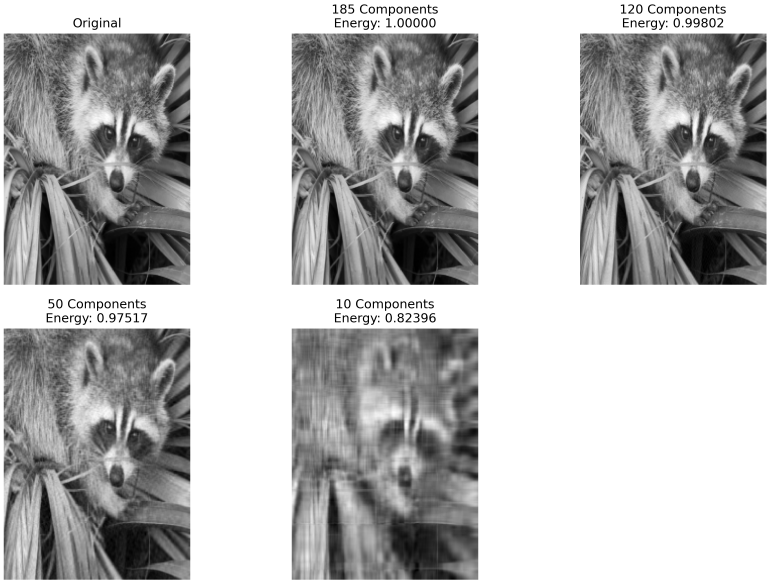
  – This data appears linear and exhibits similar scale.

- There are a few data points that are far from the main cluster of data, particularly on the upper right side of the plot. These may be outliers that could potentially influence the PCA.
- The longer red line points in the direction of the first principal component. This is the direction of greatest variance in the dataset. In PCA, this component would capture the most information about the data's structure. The shorter red line is the second principal component, it is orthogonal (at a right angle) to the first. It accounts for the next highest variance in the dataset and is uncorrelated with the first component.
- The PCA seems to be centered around the mean (not necessarily at zero), suggesting that the data has been mean-centered before the PCA was performed.

- **Part 2:**
  In Part 2, we analyze an image from the SciPy library. Due to a deprecation warning in SciPy v1.10.0 regarding `scipy.misc.face`, we have utilized `scipy.datasets.face` as an alternative. This function retrieves the classic 'face' image, which we use for PCA reconstruction.



(a) Original Image (Before Resizing)



(b) Image Reconstruction Using PCA

Figure 2: PCA on Image Data

The image dimensions are adjusted to a width of 249 pixels (x-axis) and a height of 185 pixels (y-

axis), aligning with the conventional Cartesian coordinate system where 'x' represents the horizontal axis (number of columns) and 'y' represents the vertical axis (number of rows). We then apply Principal Component Analysis (PCA) to this resized image and reconstruct it using varying numbers of principal components. This process allows us to examine the impact of different component counts on the quality of the reconstructed image.

In Figure 2, the original image is displayed alongside its PCA reconstructions with varying numbers of components. As the number of components decreases, the clarity of the reconstruction correspondingly diminishes. Retaining all 185 components, which match the image's column count, maintains most of the image's variance. Analysis of the variance captured by different numbers of components reveals that 120 components account for 99.802% of the variance, 50 components for 97.517%, and 10 components for 82.396%.

Information loss becomes visibly evident when the number of components falls well below the total column count of 185. Particularly, with 50 components, where 97.517% of the variance is explained, the degradation in detail and sharpness starts to be perceptible.

The truncation's energy loss is less than 1% when 120 components are used since they explain 99.802% of the variance, indicating a minimal remainder of energy loss.

What we learned from the results of PCA.

- This illustrates the inherent compromise in PCA, there's a balance between the accuracy of the reconstructed data and the number of principal components used. A reduction in components leads to a more abstract and less detailed reconstruction, trading precision for reduced dimensionality. Comparing reconstructions with varying numbers of PCs can aid in grasping the data's complexity and in determining the necessary number of PCs to capture the image(data)'s essential patterns.

- **Part 3:**
  Part 3 involves analyzing the trajectory data of 15 people. We specifically focus on visualizing the paths of the first two pedestrians in a 2D space.

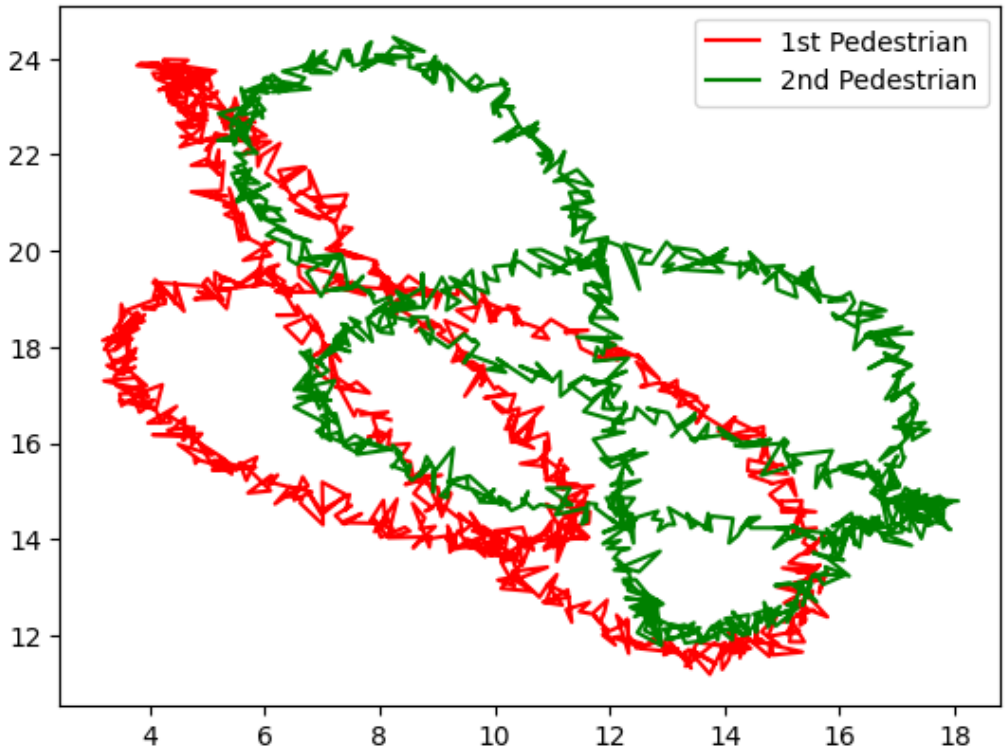
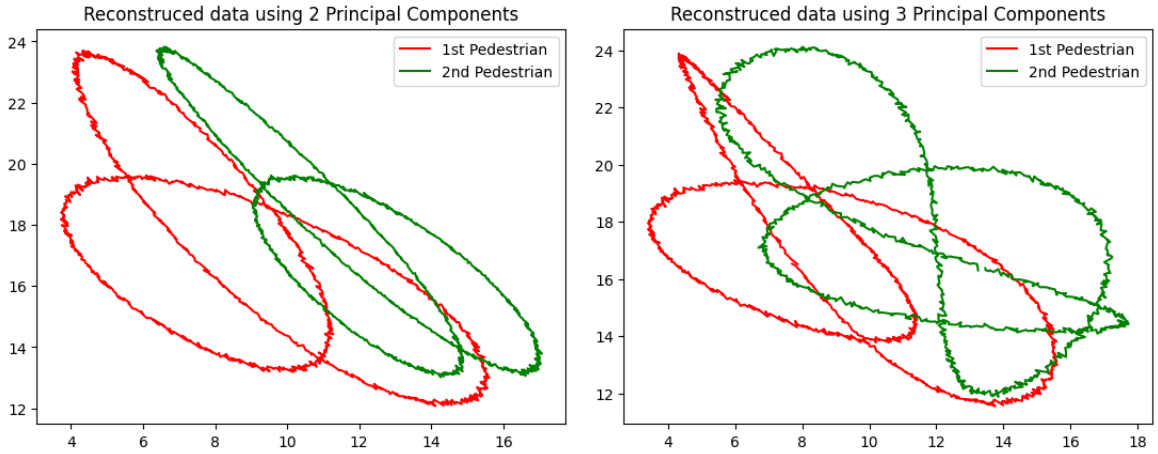


Figure 3: Original Trajectory of Pedestrian 1,2

The trajectories in figure 3 exhibit circular movement patterns and show the raw tracking data, which is replete with noise and jagged lines, capturing all the variance but lacking the detail needed to fully describe the complexity of the pedestrians' paths.



(a) Trajectory of Pedestrian 1,2 with 2 components (b) Trajectory of Pedestrian 1,2 with 3 component

Figure 4: PCA on Pedestrian Trajectories

In figure 4a, PCA with 2 components is shown. The trajectories are still distinguishable but have lost some detail compared to the multi-component reconstruction. The paths are broader and less precise, indicating that the third principal component may have contained information that contributed to a more nuanced description of the motion. The variance explained by each component is [0.47320404, 0.37601352], with an overall explained variance of approximately 0.849. Given the complex nature of the trajectories, two components are insufficient to capture the majority of the dataset's energy. The explained variance with 2 components is below 90%.

Consequently, we are considering using more than 2 components, specifically 3 components, as depicted in figure 4b. The circular patterns in the data introduce complexity that requires additional dimensions for an accurate representation. To capture more than 90% of the variance, at least 3 components are necessary, As a result, the 3-component plot shows a fairly smooth and well-defined trajectory for both pedestrians. The use of three principal components suggests that this is a higher-fidelity reconstruction of their movement, retaining more detail of the motion paths. This is also evidenced by the increase in explained variance to approximately 0.997 with 3 components. the variance explained is [0.47320404, 0.37601352, 0.14791464], resulting in an overall explained variance of approximately 0.997.

What we learned from the results of PCA.

- PCA is used to reduce the dimensionality of data, which could be very high-dimensional if it includes many tracked points or is sampled at a high frequency. This leads to noise filtering, as higher-order components may correspond to less significant movements or noise. By utilizing fewer principal components, the data can be compressed, retaining the most critical information which might suffice for some applications such as basic movement analysis or pattern recognition.

The development and implementation of the PCA code took approximately 2 hours, with a significant portion of this time dedicated to the intricacies of PCA itself. The volumne of data itself was relatively small, so implementing it did not take significant time.

Accuracy for model is commonly evaluated using metrics such as Mean Squared Error (MSE). However, the nature of our datasets doesn't comply with model accuracy. For the first dataset, which is two-dimensional, employing PCA does not reduce dimensionality but rather linearizes the data. Given its limited variable count, conventional model evaluation techniques are not applicable. The second dataset, being an image, and the third dataset, consisting of pedestrian's trajectory data, also have characteristics that limit the suitability of standard modeling approaches.

Therefore, we have opted to consider the 'energy' retained in the principal components as an alternative measure of accuracy for these datasets. It's essential to understand that while this method offers insight into the variance captured by PCA, it does not provide a comprehensive assessment of model accuracy in the same manner as error metrics like MSE. Moreover, it is important to recognize that applying PCA does not necessarily improve MSE or other error metrics, as the effectiveness of PCA largely depends on the specific characteristics and requirements of the data being analyzed.

Report on task 2, Diffusion Maps

For the implementation of Diffusion Maps, we basically followed the steps given in the exercise sheet and formed the function called `diffusion_map`. Normally ϵ is fixed as 0.05 of maximum distance. In our implementation, it is the default value however it can be changed via a parameter used in the function call. Besides the ϵ, we also wanted `data` and `L` as parameters. We used `scipy.spatial.distance` to find the maximum difference and obtain the diameter. At the end, eigenvalue and eigenvector (eigenfunction) are returned.

For the implementation and testing, we spent approximately one and a half days.

Part one: For this task, we first generated the dataset of $N = 1000$ points given by the formula in the sheet using the function `create_dataset_subtask1` which returns the value of x_k and t_k. The dataset is visualized in Figure 5.
The five eigenfunctions ϕ_l associated to the largest eigenvalues λ_l are plotted against t_k in the figure 6.

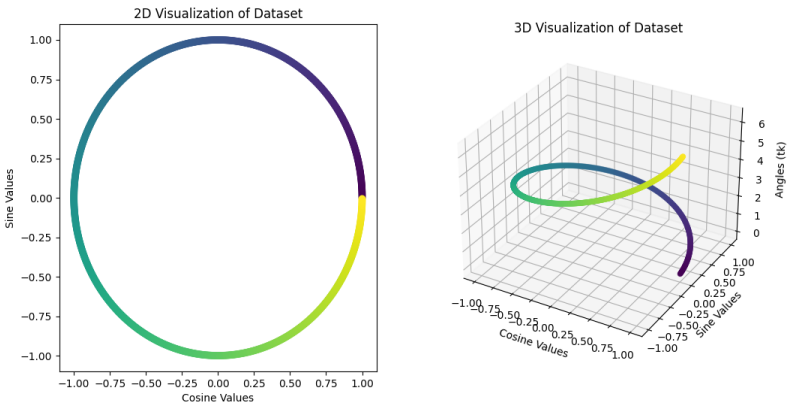


Figure 5: Dataset of 1000 points

Bonus: On inspecting the plots, we can see that the first eigenfunction is constant. This eigenfunction corresponds to the eigenvalue with the largest magnitude and likely captures that overall trend or structure of the data which is similar to Fourier analysis, where the constant eigenfunctions resembles the main part of the signal. The subsequent eigenfunctions are sinusoidal which can be compared with the sinusoidal components in Fourier Analysis.

Part 2: The Swiss roll dataset exemplifies a 2-dimensional (2D) manifold that is intricately rolled or curved within a 3-dimensional (3D) space. In general, a d-dimensional manifold can be understood as a subset within an n-dimensional space, where n is greater than d. This manifold locally resembles a d-dimensional hyperplane. In the case of the Swiss roll, the dataset fundamentally represents a 2D plane, yet it exhibits a complex, rolled structure in 3D space. This characteristic makes it a particularly interesting case for studying dimensionality reduction techniques and manifold learning. We used the `make_swiss_roll()` function of the `sklearn.datasets` library to create a Swiss roll of 5000 points as shown in Figure 7.

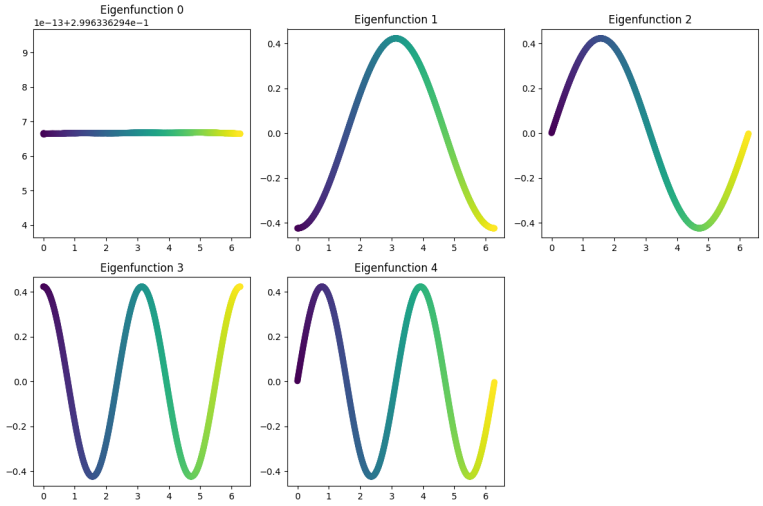


Figure 6: Eigenfunctions associated to five largest eigenfunctions

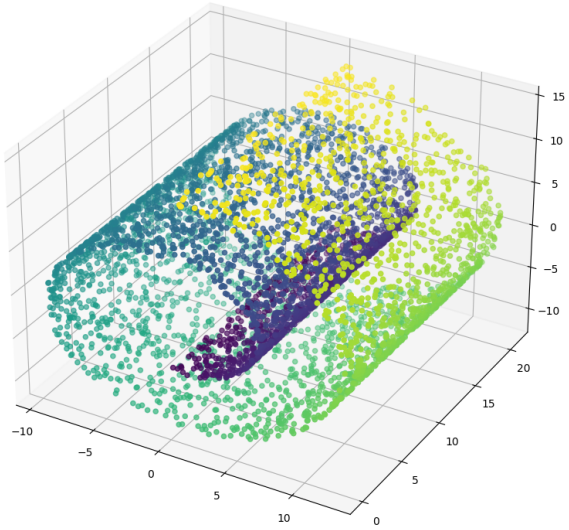


Figure 7: Swiss roll with 5000 datapoints

We used the implemented `Diffusion Map` algorithm to obtain approximations of the eigenfunctions associated with the swiss roll. We then plotted the first non-constant eigenfunction ϕ_1 against the other eigenfunctions in 2D as shown in figure 8. Inspecting the plots we can see that the ϕ_0 value is constant with respect to ϕ_1. This aligns with the nature of the eigenfunctions where ϕ_0 often represents the constant term capturing the overall trend or behaviour of the data. The values ϕ_2,ϕ_3, and ϕ_4 appear to be dependent on ϕ_1 as can be seen by the patterns in their plots. However, if we look at the plot of ϕ_5, it seems to be independent of ϕ_1. This functional independence of ϕ_l(l¿=5) shows that these eigenfunctions capture more localized and finer details that are not strongly influenced by ϕ_1. They become increasingly independent and orthogonal to lower-order eigenfunctions.

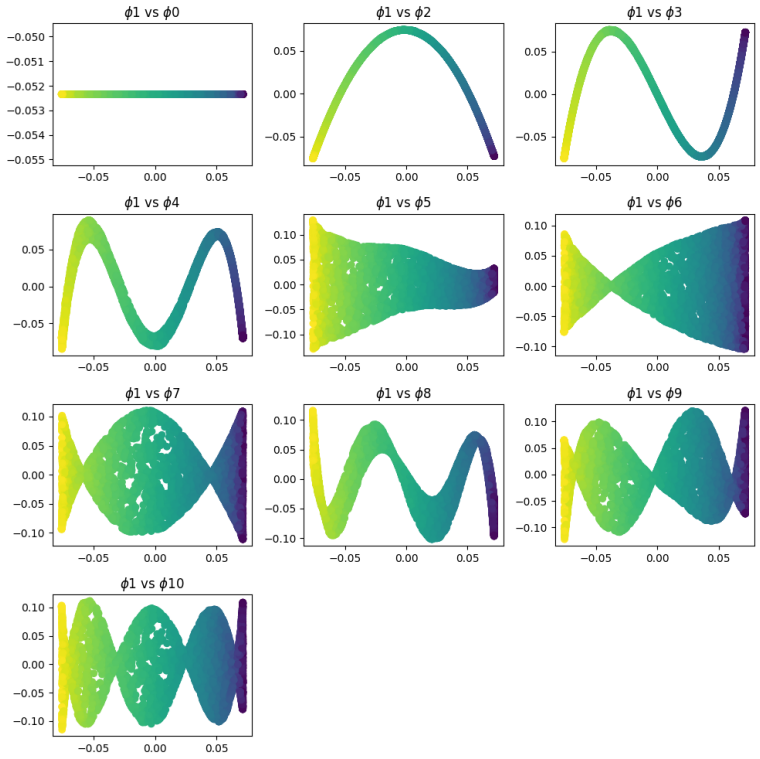
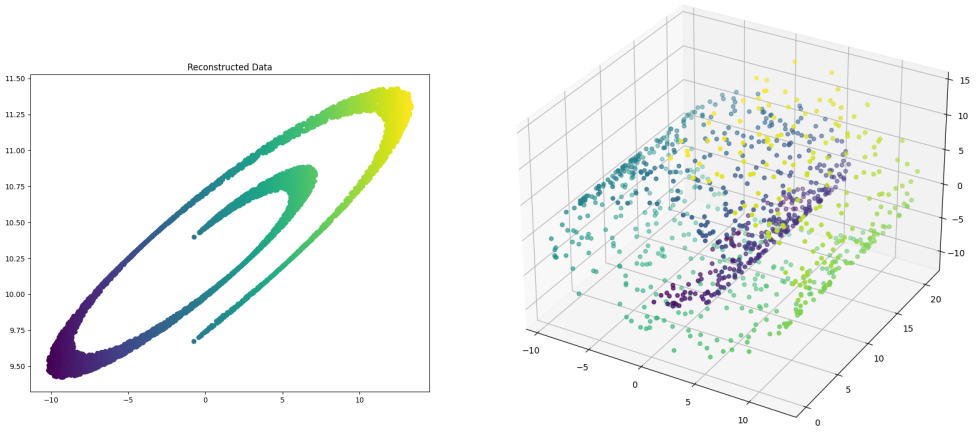


Figure 8: ϕ_1 vs other ϕ values

PCA task: Using the `PCA` implementation in task 1, we are now supposed to find the three principal components of the swiss-roll dataset. We can see that the `PCA Energy` is 100% for this, meaning that we successfully captured the entirety of the dataset's information. However, if we investigate the `PCA Energy` related with only two principal components, we can observe a substantial reduction of nearly `29%`. This reduction can be clearly seen in the plot of the reconstructed data as shown in figure 9a. This indicates the loss in variance due to the absence of the third principal component which contributes vital information and omitting it leads to a significant loss of information and an inability to represent the data.



(a) Reconstructed data with 2 PC

(b) Swiss Roll with 1000 points

Figure 9

The next experiment was to reduce the number of data points from 5000 to 1000, as shown in figure 9b. The dataset was again generated using the `make_swiss_roll` function. This dataset is much more sparse and less representative of the manifold. On repeating the experiment of plotting eigenfunctions calculated using *Diffusion Maps*, we can clearly see in figure 10 that they are much more noisy. This sparser dataset lacks the coverage required to represent the Swiss roll's complex geometry.

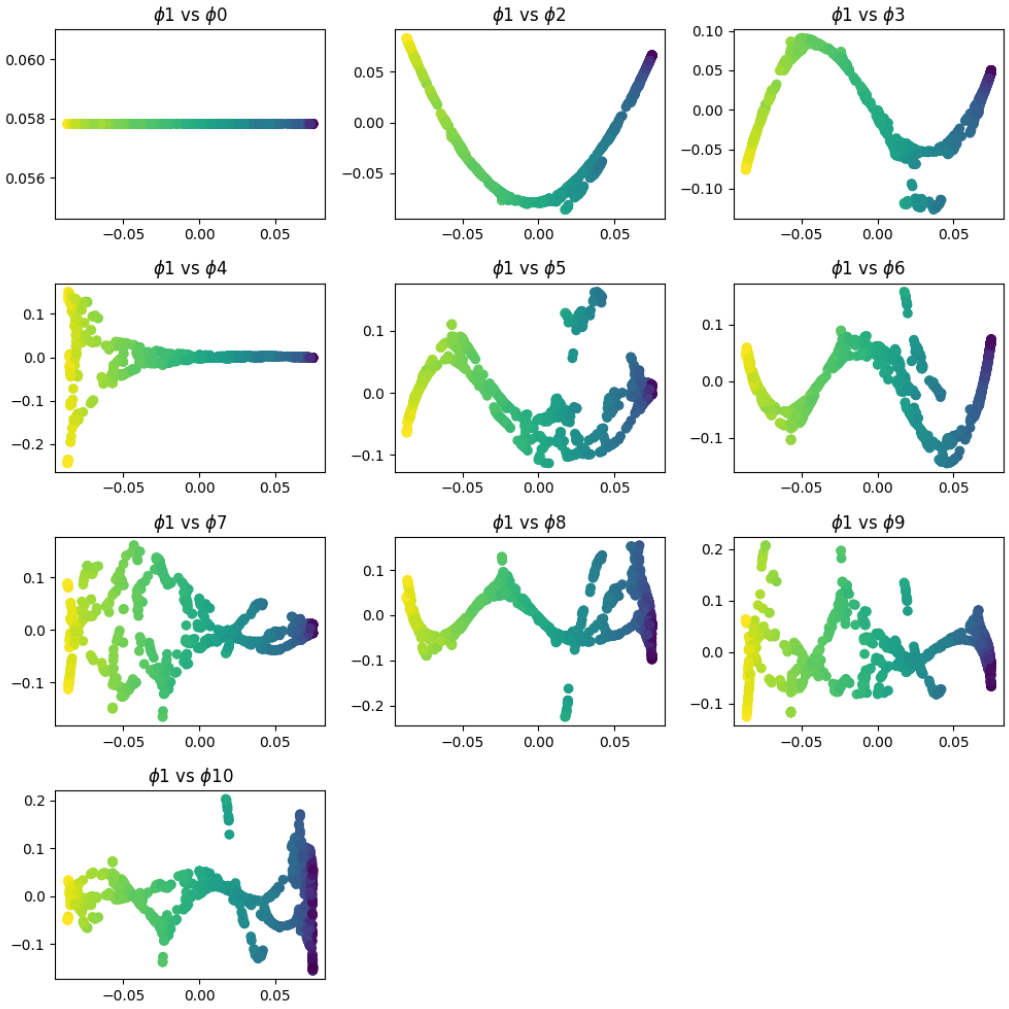
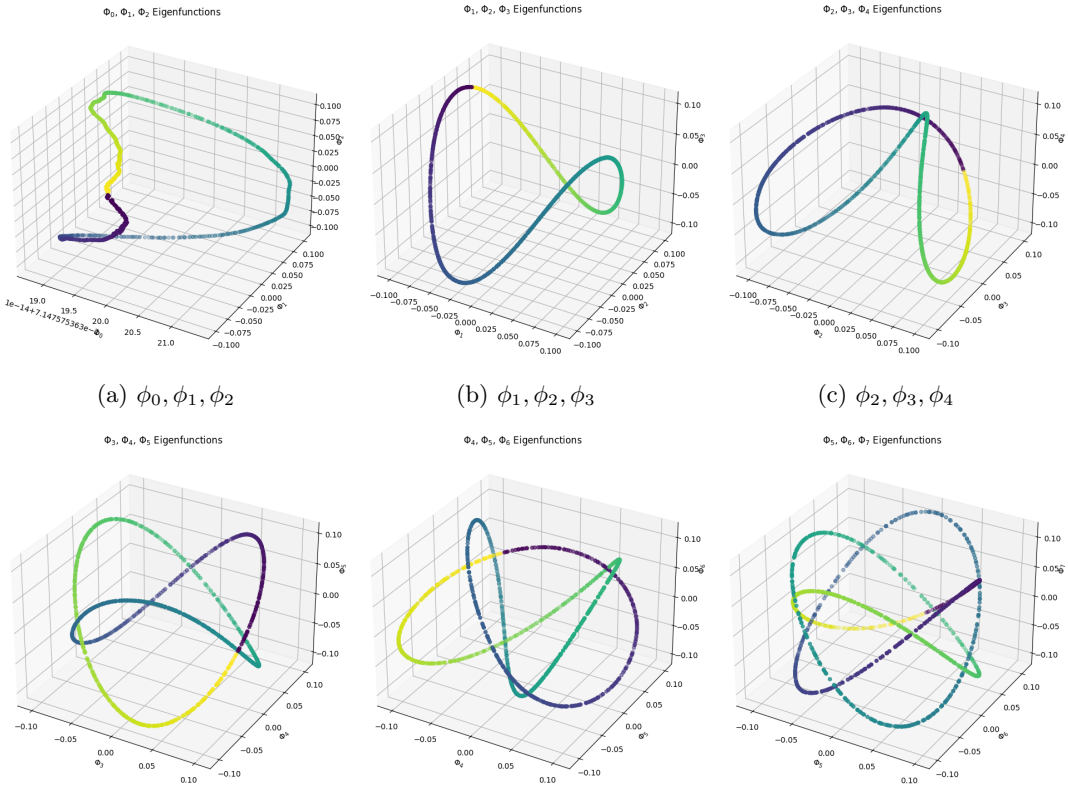


Figure 10: ϕ_1 vs other ϕ values

Part 3: As described in the sheet, Diffusion Maps do not have the same exact energy interpretation of the eigenvalues (singular values) as PCA. Therefore, we started to check 3D graphs which consist of 3 consecutive eigenfunctions. It can be observed that for the 3D graph of Φ_1, Φ_2, Φ_3, eigenfunctions can be used to fully unfold the data, which is not the case for the graph of the other triples, as can be seen in figure 11.

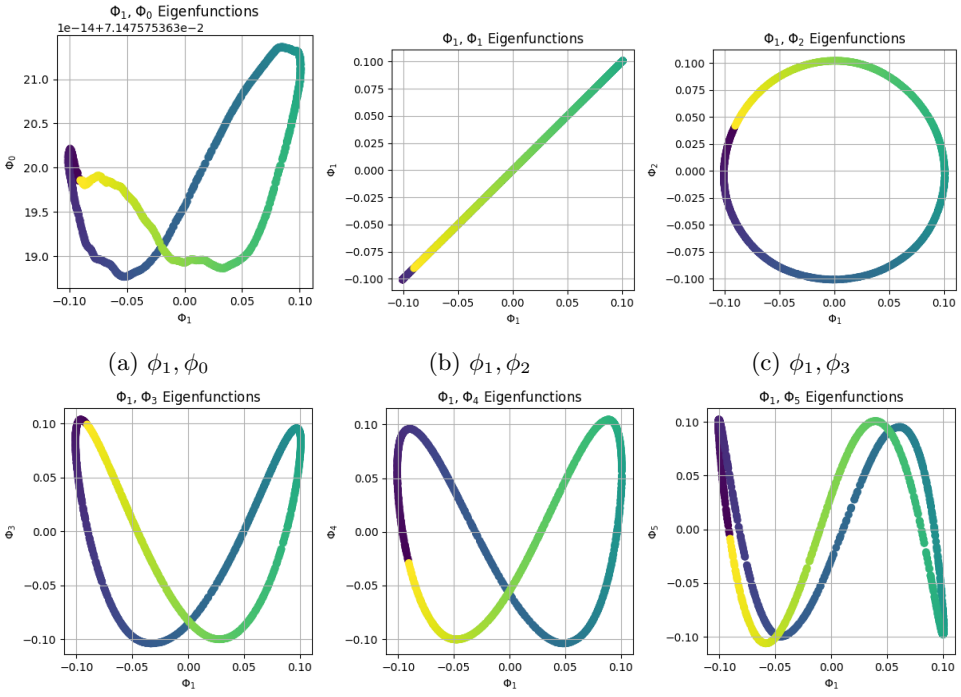
If we further dig inside the graph of 2D consecutive eigenfunctions, we can observe that the graph of Φ_1, Φ_2 eigenfunctions do not intersect. Hence it would be fair to say it is a proper embedding. If we check the other 2D graphs in figure 12, they all contain intersections in some ways. As Φ_1, Φ_1 is also comparison with itself, we cannot consider that.



(a) ϕ_0, ϕ_1, ϕ_2 (b) ϕ_1, ϕ_2, ϕ_3 (c) ϕ_2, ϕ_3, ϕ_4

(d) ϕ_3, ϕ_4, ϕ_5 (e) ϕ_4, ϕ_5, ϕ_6 (f) ϕ_5, ϕ_6, ϕ_7

Figure 11: Graphs for 3 Eigenfunctions in Euclidean Space



(a) ϕ_1, ϕ_0 (b) ϕ_1, ϕ_2 (c) ϕ_1, ϕ_3

(d) ϕ_1, ϕ_4 (e) ϕ_1, ϕ_5 (f) ϕ_1, ϕ_6

Figure 12: Graphs for 2 Eigenfunctions in 2D Plane

Bonus task: For this task, we are required to download the `datafold` library, which speacialize in manifold learning, and repeat the experiment on swiss roll dataset. The Swiss roll dataset has a very unique structure and lies on a curved manifold, therefore, PCA may not be able to provide an adequate low-dimensional representation. We followed the steps in the provided tutorial for s-curve manifold to compute and visualize the eigenvectors of the swiss roll. The code provided in the tutorial made it very easy to modify for swiss roll. The outputs generated by using the datafold library, as seen in figure 13, are consistent with the outputs generated by us. One of the main advantages that we observed by using the Datafold library was its high level of optimization. The library provided results in significantly less time compared to our previous implementation. The library also provides the function `LocalRegressionSelection` for automatic selection of optimal lower-dimensional embeddings which is a complex task, especially for high-dimensional data.

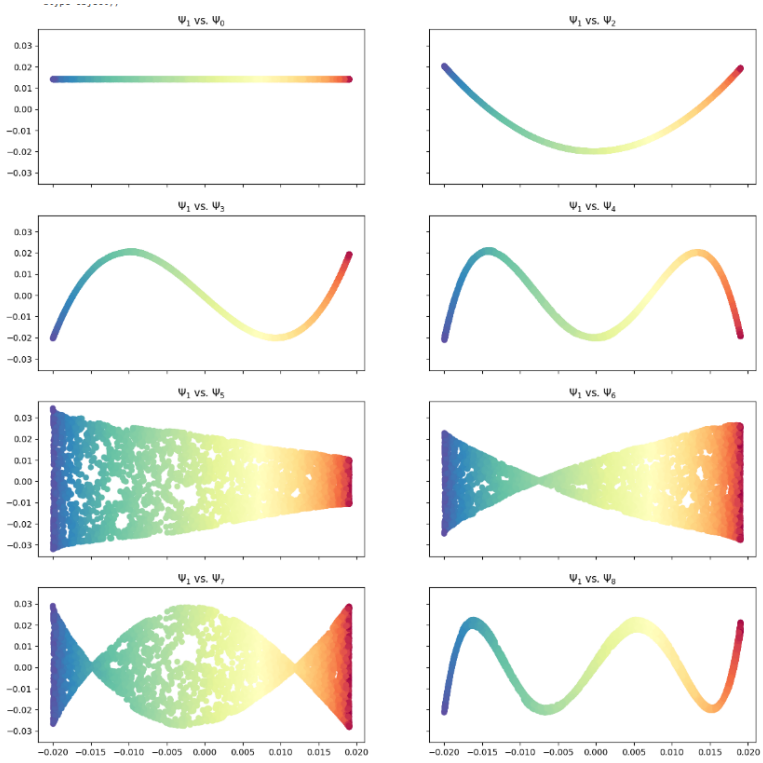


Figure 13: ϕ_1 vs other ϕ values generated using datafold

Report on task 3, Training a Variational Autoencoder on MNIST

Task 3 explores the option of using a Variational Autoencoder (VAE). In order to train our first model, we will use the MNIST dataset [2] containing 60,000 training samples as well as 10,000 test samples of handwritten numbers and normalise the pixel values between 0 and 1. The task specifies what the model should look like:

- two-dimensional latent space
- multivariate diagonal Gaussian distribution as approximate posterior $q_\phi(z|x)$
- Encoder (outputs the mean and standard deviation of $q_\phi(z|x)$):
  - 2 hidden layers with 256 units each and ReLU activation functions
  - multivariate diagonal Gaussian distribution as likelihood $p_\theta(x|z)$
- Decoder (outputs only the mean of $p_\theta(x|z)$):
  - 2 hidden layers with 256 units each and ReLU activation functions
  - standard deviation for the decoder distribution as one floating-point, trainable variable

- multivariate diagonal standard normal distribution as the prior $p(z)$

Additionally, the Adam optimiser is used with a learning rate of 0.001, as well as a batch size of 128 for training. The libraries used in our code are depicted in table 1 together with the functionalities that we utilised.

Tensorflow [1]	download the MNIST dataset & implement Early Stopping
Numpy [3]	generate random samples from a normal (Gaussian) distribution
Matplotlib [4]	visualise the constructed and generated images

Table 1: Libraries and their utilised functions for the task of VAE

1. What activation functions should be used for the mean and standard deviation of the approximate posterior and the likelihood and why?

The choice of activation functions for the mean and standard deviation in a VAE is critical due to the properties of Gaussian distributions. For the mean μ of the approximate posterior $q_\phi(z|x)$, a linear activation function is appropriate as the mean can take on any real value. Hence, no activation function, or the identity function, is typically used:

$$\mu = W_\mu h + b_\mu \tag{1}$$

where W_μ and b_μ are the weights and biases for the mean, and h is the output from the last hidden layer of the encoder.

For the standard deviation σ, we require an activation function that ensures the output is positive, as the standard deviation must be non-negative. The softplus function is commonly used for this purpose:

$$\sigma = \log(1 + \exp(W_\sigma h + b_\sigma)) \tag{2}$$

where W_σ and b_σ are the weights and biases for the standard deviation.

The softplus function is smooth and gradually increases, avoiding any abrupt changes in the gradient, which benefits the stability of the learning process. Additionally, it closely approximates a ReLU function for large input values, effectively preventing the standard deviation from collapsing to zero.

For the likelihood $p_\theta(x|z)$, the decoder outputs only the mean, and since the pixel values are normalized between 0 and 1, a sigmoid activation function is used to ensure the output is within this range:

$$\hat{x} = \sigma(W_{\hat{x}} z + b_{\hat{x}}) \tag{3}$$

where $\sigma(\cdot)$ is the sigmoid function, ensuring that the reconstructed input $\hat{x}$ is in the range $[0, 1]$, corresponding to the normalized pixel values.

2. What might be the reason if we obtain good reconstructed but bad generated digits?

There exist multiple possible reasons for the VAE to produce badly generated digits like learning rate, training duration, or loss function. However, it is apparent that the latent space of our model is very small. This can result in the model not having enough capacity to represent the diversity of the data during generation. Because of this, we will increase the latent space in part 5 of this task.

3. Train the VAE

In question 3, we train the VAE. After the 1st, 5th, 25th, and 50th epoch, we will plot the latent representation, plot reconstructed digits, and plot generated digits. Additionally, we do the same after convergence. To determine when the optimisation converges, we make use of Early Stopping which monitors `val_loss` and uses patience of 5. The exact point of convergence will differ since the VAE does not work deterministically. In the figures we provided in the report, Early Stopping happened after epoch 55.

3.1. Plot the latent representation

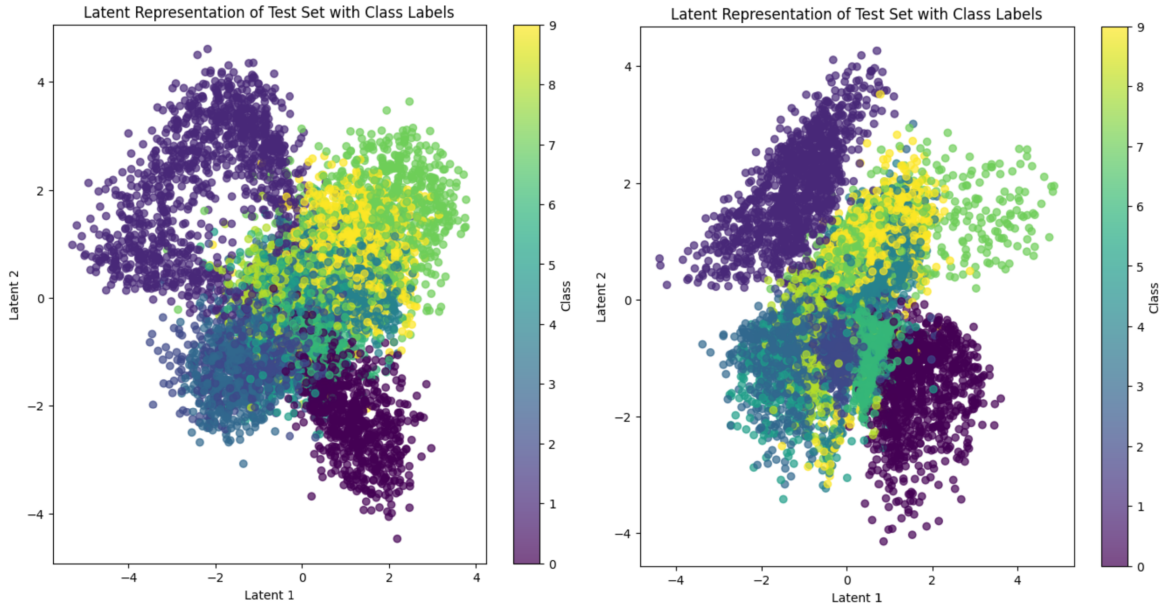
We have our results of transformation of the latent space of a Variational Autoencoder (VAE) as it is trained over multiple epochs. The plots show the latent representations of a test dataset, where each point corresponds to a high-dimensional data instance compressed into two latent dimensions for visualization purposes. The color indicates class labels, which are not used during the unsupervised training of the VAE but are shown here to demonstrate the clustering capability of the model.

Subplot 14a depicts the latent space after 1 epoch of training. The data points are largely overlapped, indicating that the VAE has not yet learned to separate the different classes effectively in the latent space.

Subplot 14b shows the latent space after 5 epochs. The beginnings of cluster formation can be observed, with the VAE starting to distinguish between classes, although considerable overlap still exists.

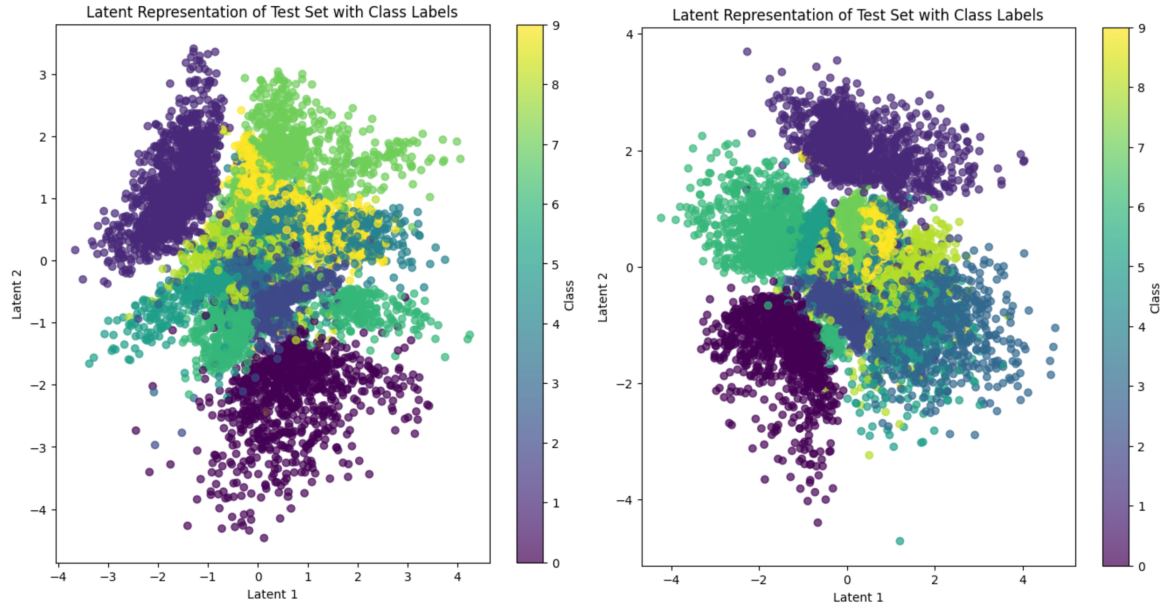
Subplot 14c represents the latent space after 25 epochs. The clusters are more defined, showing that the VAE has improved in mapping similar data points (from the same class) closer together in the latent space while separating different classes.

Subplot 14d presents the latent space after 50 epochs. The clusters are well-defined and mostly non-overlapping, illustrating that the VAE is now capable of organizing the data in a way that similar instances are located in proximity to each other, forming distinct clusters corresponding to the different classes. This suggests a matured learning where the encoder part of the VAE has developed a more structured understanding of the data.



(a) 1 epoch

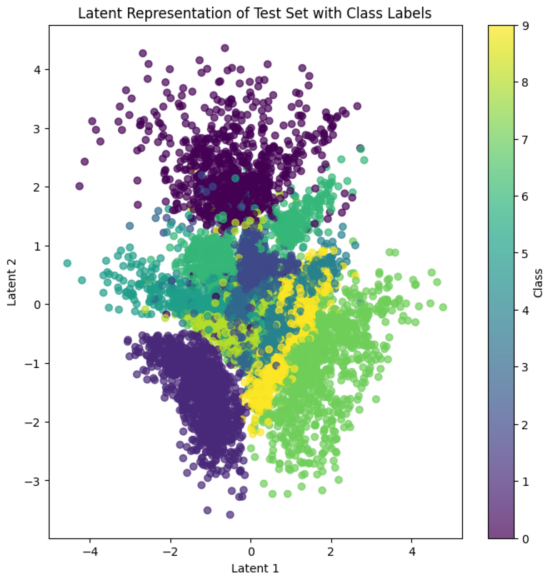
(b) 5 epochs



(c) 25 epochs

(d) 50 epochs

Figure 14: Latent Representation after different numbers of epochs



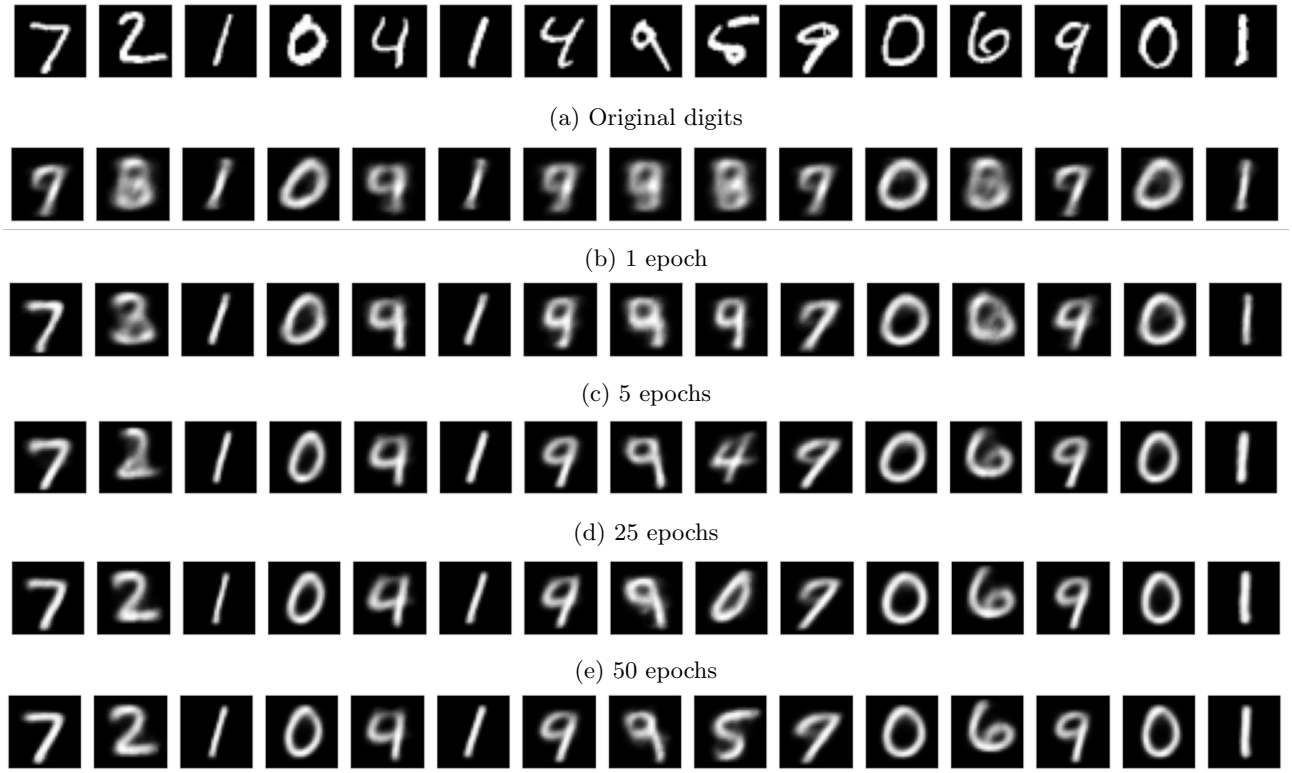
(e) after convergence

Figure 14: Latent Representation after different numbers of epochs

3.2. Plot 15 reconstructed digits and the corresponding original ones.

To generate the reconstructed digits, we use $vae.predict(x\_test)$. Plotting them creates figure 15, the subfigures show different training times. After only 1 (15b) and 5 (15c) epochs, it is difficult to achieve proper results for all digits. Increasing the training time to 25 (15d) or 50 (15e) epochs helps to recognize most of the digits and only shows a small difference between both training times.

Overall, the model works well and 12 of the reconstructed digits depict the correct number for 25 and 50 epochs. Two out of the three errors result from the same digits: A 4 turns into a 9. This mistake is easy to retrace. The two lines on top of the 4 just need to be connected with a slight arc to change the number accordingly. The third error recreates a 4 for 25 epochs, even though the original digit was a 5. This mistake seems unnecessary since multiple lines do not align. For 50 epochs the same digit is problematic and shows a 0. An explanation for the mistakes on this digit is the sloppy handwriting which is distinctly different to a properly written 5. After convergence, even this digit is correctly displayed in the reconstruction. All of the reconstructed digits display a blurriness to them which is to be expected. However, there are methods to reduce blurriness, which again include a larger latent space.



(a) Original digits

(b) 1 epoch

(c) 5 epochs

(d) 25 epochs

(e) 50 epochs

(f) after convergence

Figure 15: Reconstructed digits after different numbers of epochs

3.3. Plot 15 generated digits, i.e., decode 15 samples from the prior.

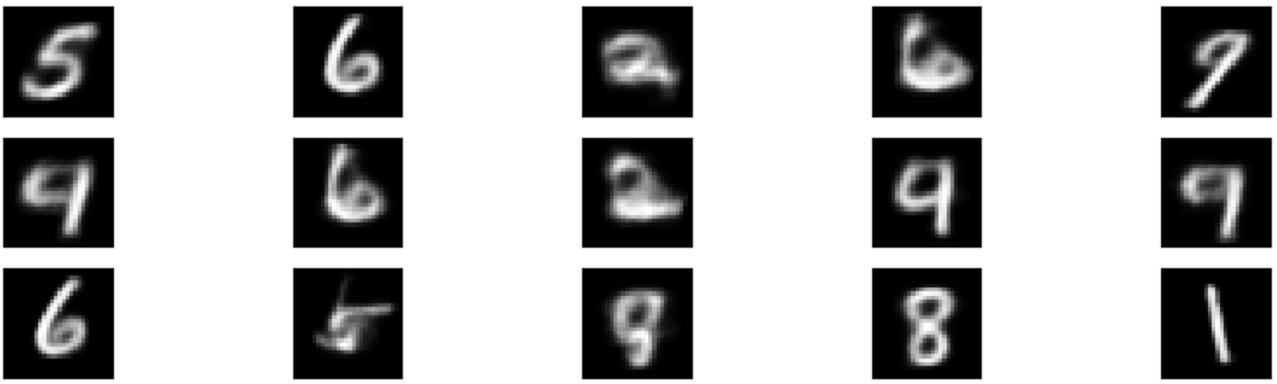
To generate new digits, we use *decoder.predict*(*random_samples*). Plotting them creates figure 16, the subfigures show different training times. After 1 epoch (16a), barely any digits can be recognized. This changes for 5 (16b) and 25 epochs (16c). For 50 epochs (16d) the identifiability decreases again slightly.

Most of the generated digits in figure 16c can be recognized as a specific number. The first generated digit could be interpreted either as 1 or as 7. Since the original images showing the number 1 however only consist of a line, the first generated digit most likely corresponds to the number 7. The second digit shows the number 8 with the left side less pigmented making it similar to the number 3. The third digit depicts a problem we already encountered in the reconstructed digits: Numbers 4 and 9 look similar. Both 4 and 9 could be interpreted with a preference towards the number 9. The second digit in the second row is the only image that cannot certainly be assigned a specific number. The remaining digits are easy to recognize and assign.

(a) 1 epoch

(b) 5 epochs

(c) 25 epochs



(d) 50 epochs

Figure 16: Generated digits after different numbers of epochs

(e) after convergence

Figure 16: Generated digits after different numbers of epochs

4. Plot the loss curve

ELBO loss stands for 'evidence lower bound loss' and is commonly used in VAEs. Figure 17 shows the loss development over the epochs. In the first two epochs, the training loss drops quickly to a value of 165.5712, the test loss being slightly lower at 162.4413. After that, the loss decreases slowly until we achieve a training loss of 141.2879 and a test loss of 141.8026. This behavior is what was expected. The values of training and test loss leave a small generalization gap, meaning that our model is able to perform well on unseen data as well. Since the training loss curve is still going down when we stop training, the model is not overfitting and we could possibly increase the training time (amount of epochs) to achieve the best possible outcome given the same parameters.

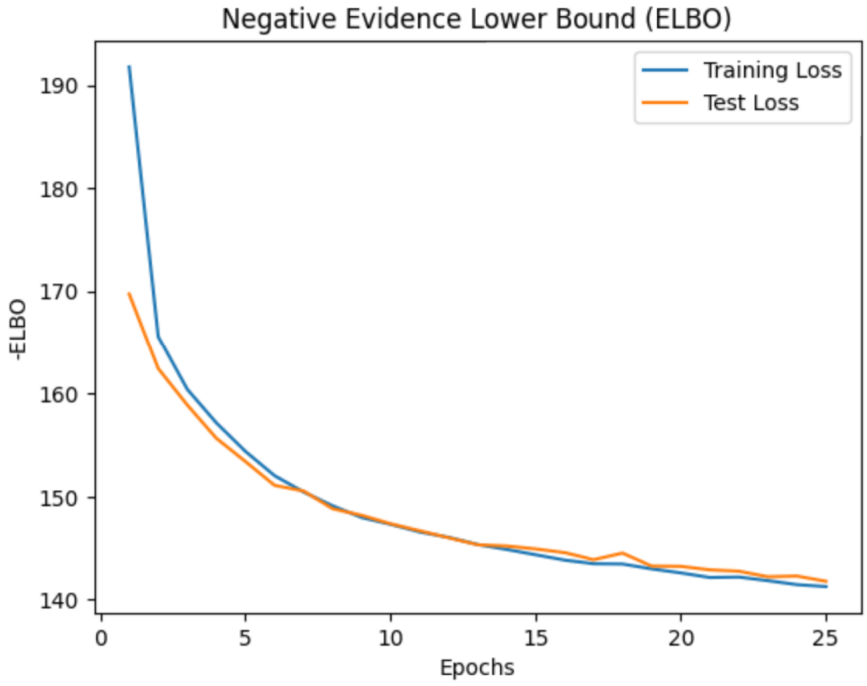


Figure 17: ELBO Loss vs. epochs on 2-dimensional latent space

5. Train the VAE using a 32-dimensional latent space

Despite our assumption, increasing the latent space does not generate better digits with a training time of 25 epochs. In figure 18, the depicted digits are less recognizable, less than half of them can be assigned a certain number.

Figure 18: Generated digits on 32-dimensional latent space

At first glance, figure 19 looks similar to the previously generated loss curve. Once we take a closer look, we notice that increasing the latent space to 32 has a positive effect on the loss: After just two epochs, the loss already dropped to 129.1468 for the training and 120.9672 for the test, both values are lower than the previous loss after our full training time. In comparison to the last test loss, it is more stable now and does not depict any spikes in its curve. After the 25th epoch, we reach a training loss of 101.1582 and a test loss of 101.0482 which is significantly better with a difference of roughly 40 each. Again, we can see that the training loss is still decreasing in the end. This time, however, the decrease involves only decimals.

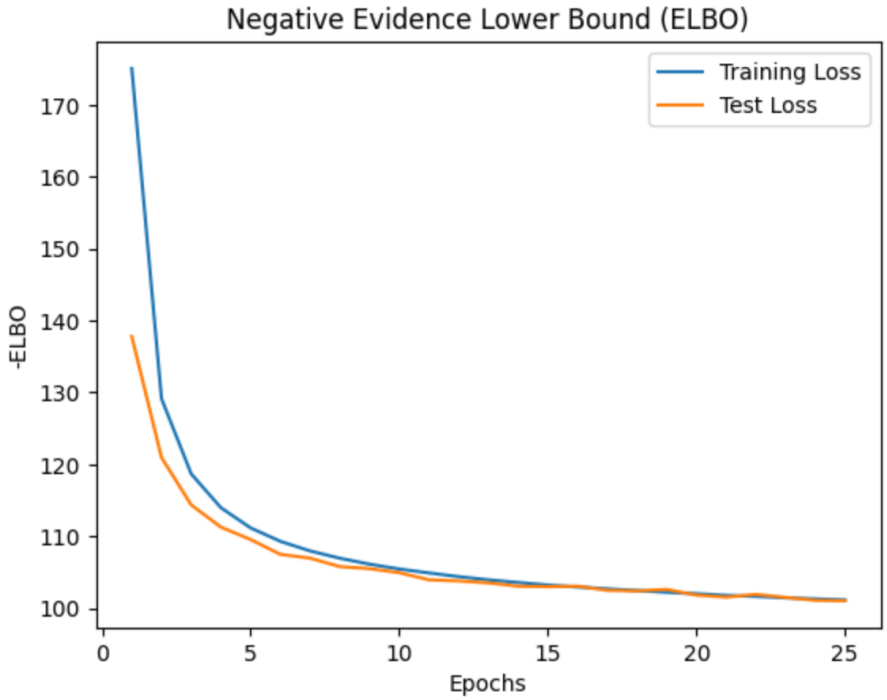


Figure 19: ELBO Loss vs. epochs on 32-dimensional latent space

Approximately 15hours have been needed in order to implement and test the VAE model itself.

The observations made in 5. prove that simply increasing the latent space of a model doesn't necessarily improve its performance and it can be explained by the fact that the model isn't complex enough to deal with the added capacity of representation brought by the larger latent space and/or by the fact that the dataset's complexity (here the MNIST) doesn't warrant a larger latent space as it has a low intrinsic dimensionality itself.

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

Task 4 is centered around reconsidering the current fire evacuation plan for the MI building at the Technical University of Munich. Initially, we download the FireEvac dataset, provided as Numpy [3] files. Train and test data are loaded into Jupyter Notebook [5] with the command `np.load`. Once the data was split into x and y values, we can visualise it with a scatter plot, as can be seen in figure 20. The left subfigure displays the training data which holds 3000 x-y-positions and thus has more dense positions than the test data on the right with only 600 x-y-positions. Both plots show the possible positions of pedestrians and outline the form of the MI building.

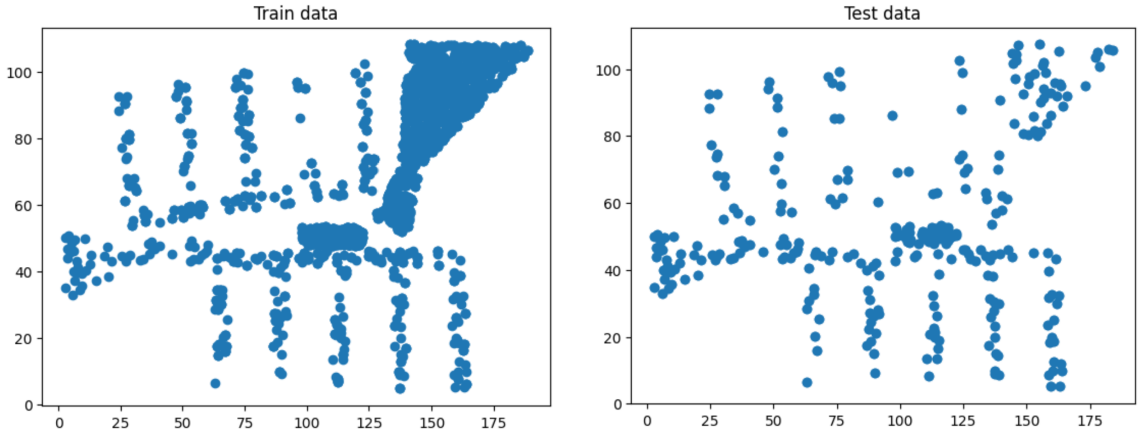


Figure 20: Scatter Plot of training and test data before scaling

As we want to learn the distribution of people within the MI building $p(x)$, we train a VAE on the FireEvac data. The implementation remains the same as in Task 3 with the exception of a few parameters so that the model can fit the new data better.

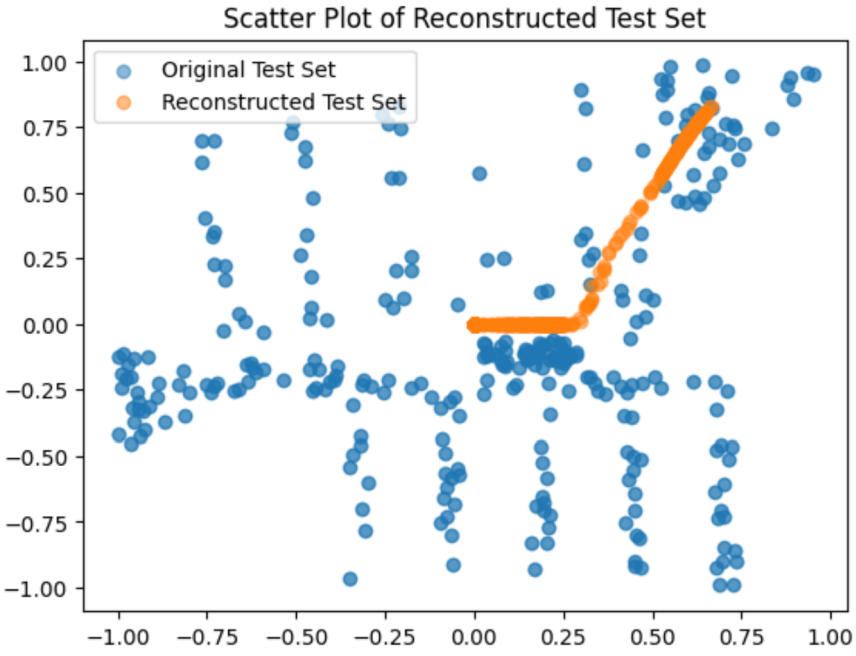


Figure 21: Comparison of the original and reconstructed test set

While the reconstructed test set in figure 21 does not represent the complete original test set well, it does visualise the entrance of the MI building. Similarly, figure 22 also depicts the entrance only. Since there is no comparison here, the image is scaled to start at 0.0.

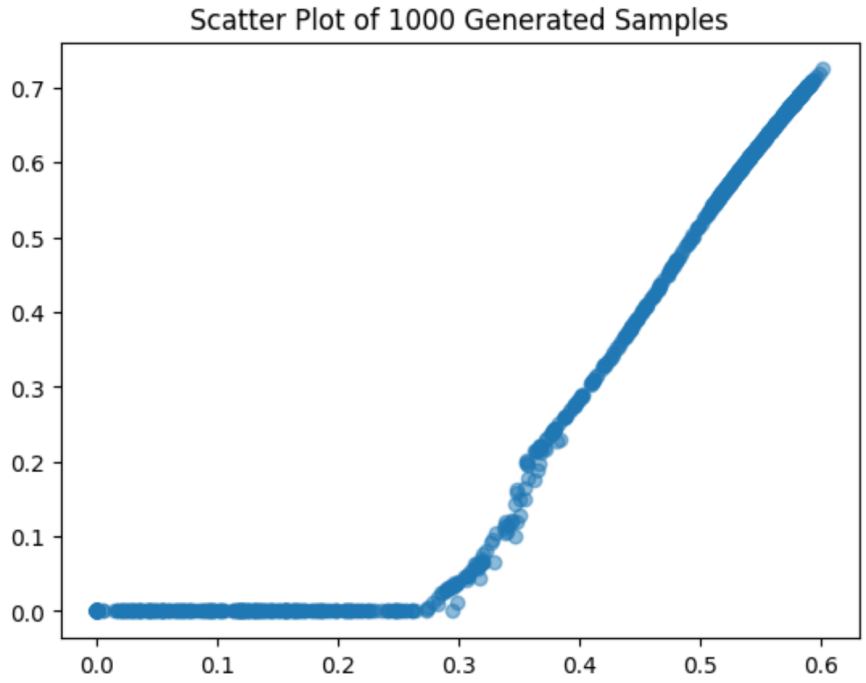


Figure 22: Generated pedestrians in the MI building

The rectangle in figure 23 defines a sensitive area in front of the main entrance where the number of people should not exceed 100. Since the generated samples follow a straight line instead of scatter in a defined area, only a few pedestrians enter the rectangle on its left top. After increasing the number of generated samples to 5000, the critical number of 100 is surpassed.

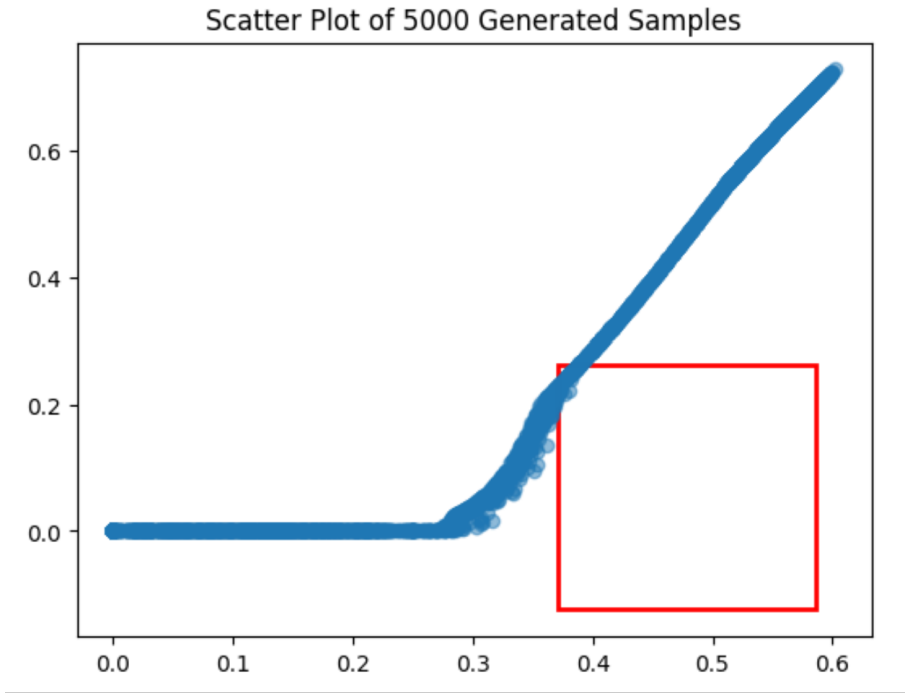


Figure 23: Generated pedestrians (blue) and the sensitive area (red)

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[3] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[4] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[5] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.