Report for exercise 2 from group J

Tasks addressed:	4
Authors:	Wenbin Hu (03779096) Yilin Tang (03755346) Mei Sun (03755382)**Team Leader** Daniel Bamberger (03712890)
Last compiled:	2023–05–29
Source code:	https://github.com/HUWENBIN2024/TUMCrowdModelingGroupJ/tree/main/ex3

The work on tasks was divided in the following way:

Wenbin Hu (03779096)	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	1/3
Yilin Tang (03755346)	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	0
Mei Sun (03755382)**Team Leader**	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	1/3
Daniel Bamberger (03712890)	Task 1	0
	Task 2	0
	Task 3	0
	Task 4	1/3

Report on task 1, Principal Component Analysis

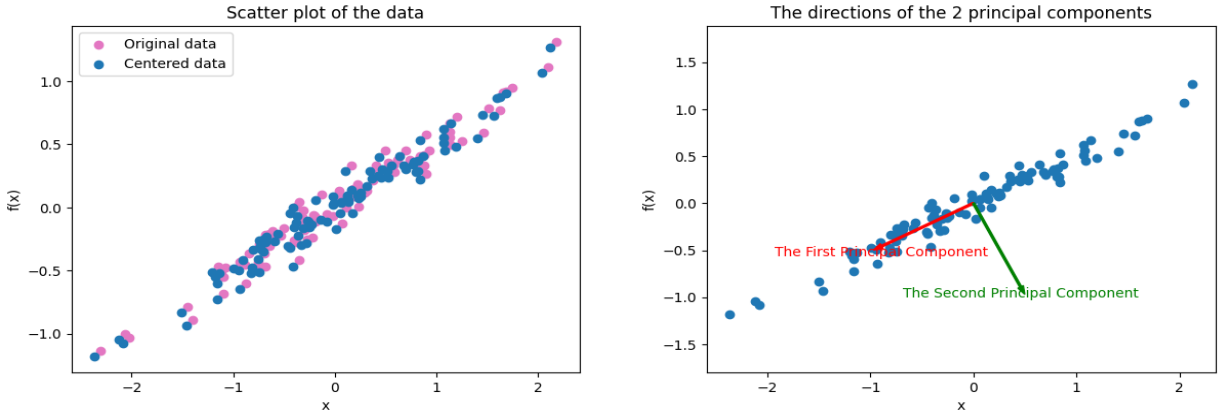
In this task, we have used a method called Principal Component Analysis (PCA) based on singular value decomposition (SVD). This method is useful in many areas of data analysis and helps to compress data effectively and efficiently. The total time estimated for implementing, testing, and reporting on this task is about 10 hours.

To measure the data, we developed our own program from scratch. This program calculates the energy loss using different numbers of principal components. The energy loss results can be visualized in a graph using a specific threshold. Additionally, we checked the quality of the reconstructed data through visualization. To evaluate the accuracy of the data reconstruction, we focused on two measures. The first is the percentage of energy captured by the principal components that were not set to zero which helped us understand how much information was retained in the reconstructed data. The second measure was the norm of the difference between the original data and the reconstructed data which allowed us to quantify the overall dissimilarity between the two. When we reconstructed the data without setting any singular value to zero, the method proved to be highly accurate. The reconstructed data closely matched the original data, indicating that the method preserved the majority of the information.

By completing this part of the task, we have gained a deeper understanding of SVD and how it is applied in various aspects of our daily lives. SVD is a common approach used to compress data and save storage space with minimal loss in data quality.

Part 1: Implement PCA for SVD and apply to dataset

The dataset "pca_dataset.txt" contains 100 data points, each consisting of 2 components. As shown in Fig.1(a), the scatter plot of this dataset visually represents the distribution of the data points. Upon performing PCA on the dataset, we found that the first principal component captures 90% of the total energy, indicating that it contains the majority of the information in the dataset. In contrast, the second principal component only captures 10% of the energy. As shown in Fig.1(b), the directions of these principal components are illustrated where the red line represents the direction of the first principal component, and the green line represents the second principal component. These directions are determined by the eigenvectors obtained from the SVD of the dataset. The SVD is utilized to calculate the matrix V, which is composed of the eigenvectors. The visual representation in Fig.1(a) further highlights the dominance of the first principal component. All data points have longer projections in the direction of the red line, indicating that this component contains the most information and contributes significantly to the dataset.



(a) (b)

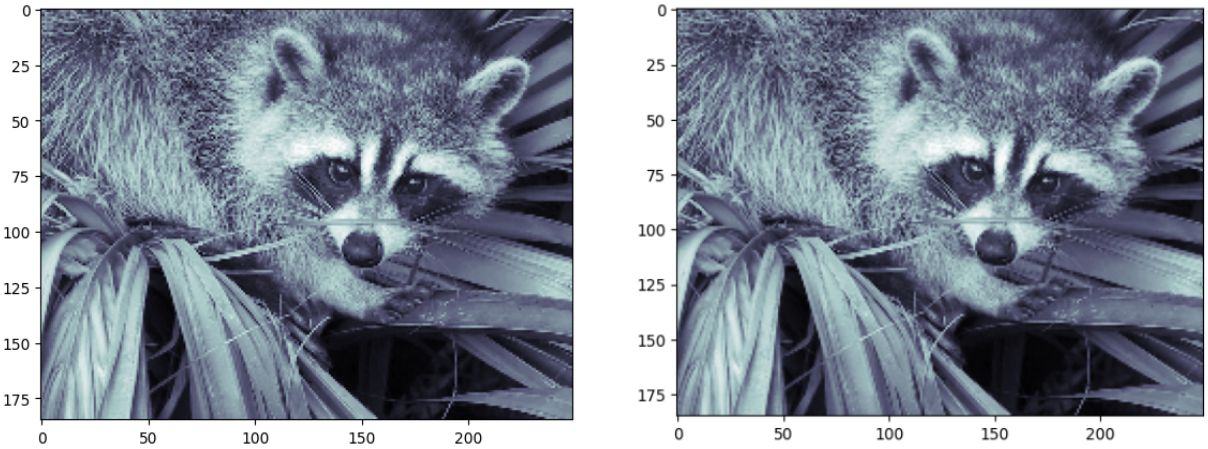
Figure 1: Scatter plot of the data contained in pca_dataset.txt.[a] plot the centered data and the original data [b] shows the plot of the centered data with the directions of the 2 principal components.

Part 2: Apply PCA to image

In this part, we utilized PCA on an image of a raccoon. The image was obtained using "scipy.misc.face" and transformed into grayscale. As shown in Fig.2(a), it was resized to a dimension of 249 × 185 pixels. To

apply PCA, we treated each column of the image as a separate data point. We then performed PCA on these data points. Our goal was to reconstruct the original image using different numbers of principal components: all, 120, 50, and 10. The results of the reconstruction process as shown below. When using all principal components, the reconstructed image closely resembles the original image. However, even with only 50 principal components, some slight differences become noticeable. It is not until we reduce the number of principal components to 10 that the differences between the original and reconstructed images become very evident. These findings highlight the importance of the number of principal components used in the reconstruction process. Fewer components result in a less accurate representation of the original image.

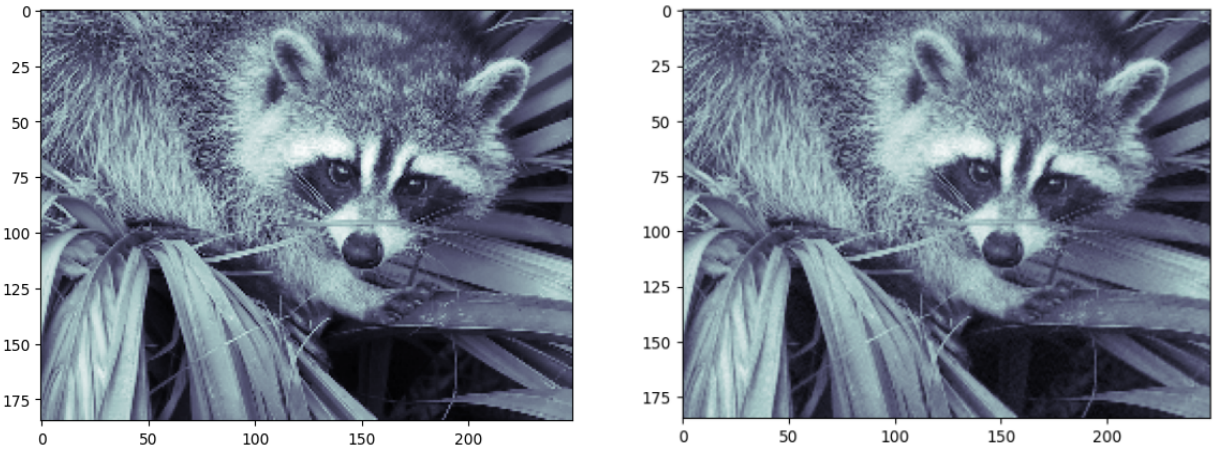
- **All principal components**
  As shown in Fig.2(b) capture 100% of the energy, loss 0% energy. It is the original image.



(a) (b)

Figure 2: [a] The original image [b] the reconstructions of the image with all principal components.

- **120 principal components**
  As shown in Fig.3(b) capture 99.04% of the energy, loss 0.96% energy. It is almost indistinguishable from the original image.

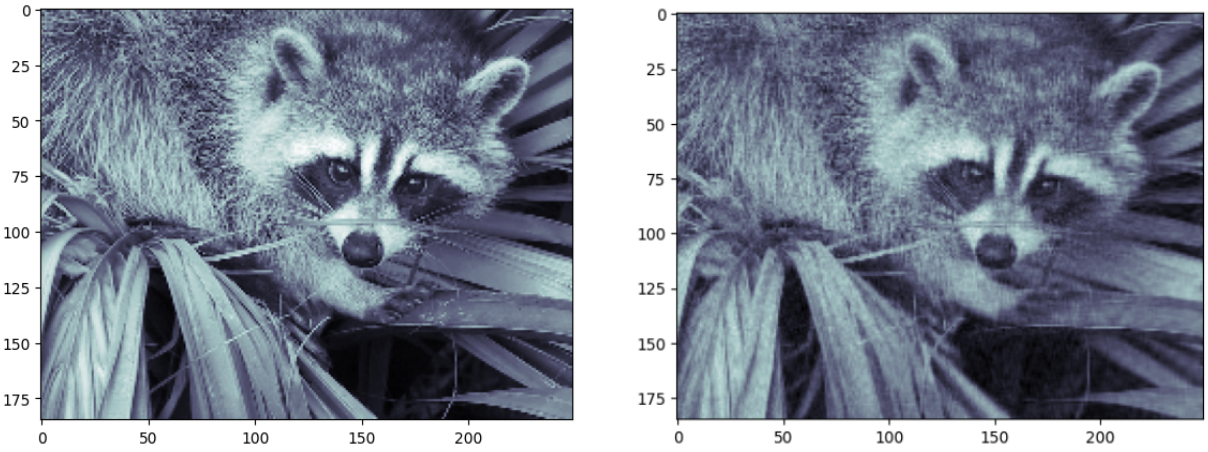


(a) (b)

Figure 3: [a] The original image [b] the reconstructions of the image with 120 principal components.

- **50 principal components**

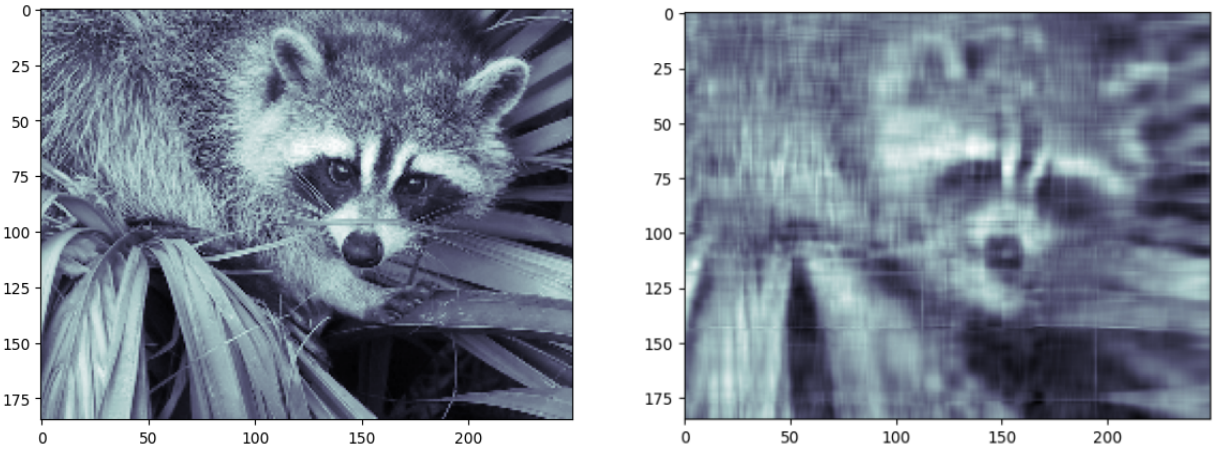
As shown in Fig.4(b) capture 91.83% of the energy, loss 8.17% energy. The information loss is visible, but the raccoon and its environment can be seen clearly.



(a) (b)

Figure 4: [a] The original image [b] the reconstructions of the image with 50 principal components.

- **10 principal components**
  As shown in Fig.5(b) capture 72.73% of the energy, loss 27.27% energy. The information loss is visible, and the image is very blurry. We only could see the profile of the raccoon and its color patches.



(a) (b)

Figure 5: [a] The original image [b] the reconstructions of the image with 10 principal components.

The information loss is visible when the principle components are less than 50. We made a conditional loop to compute how many principal components are needed to make energy lost smaller than 1%. As shown in Fig.6, we need at least 120 principal components to make energy lost smaller than 1%. 120 principal components lost 0.96% of the energy.

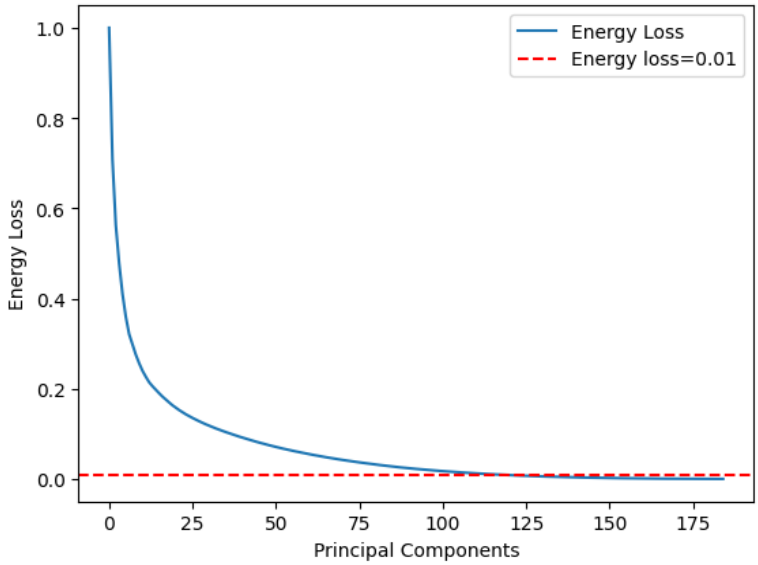
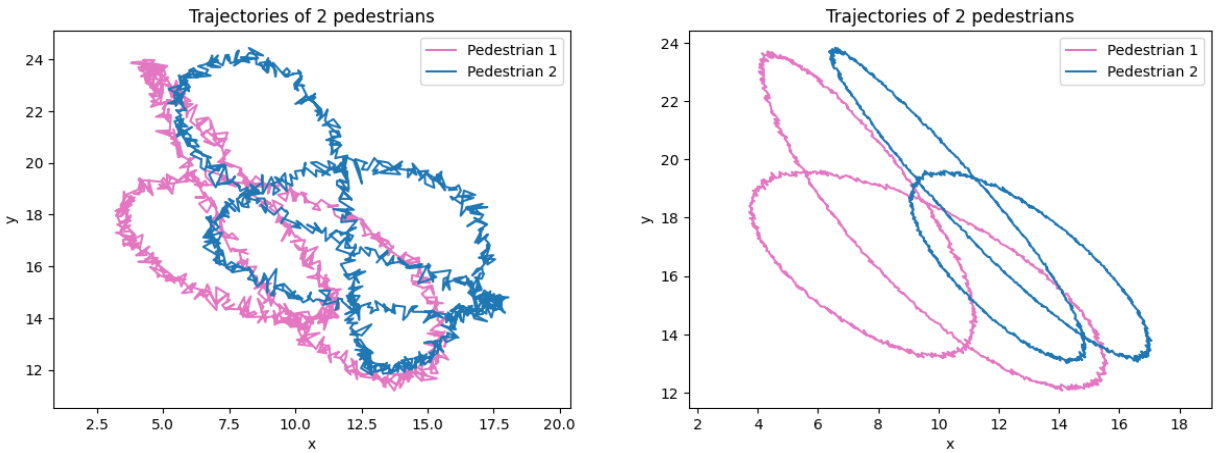


Figure 6: The energy loss with the different number of principal components.

Part 3: Apply PCA to pedestrian trajectory

In this part, we import the dataset "data_DMAP_PCA_vadere.txt". The dataset represents the positions of 15 pedestrians at different time steps. It contains 1000 data points with 30 components.

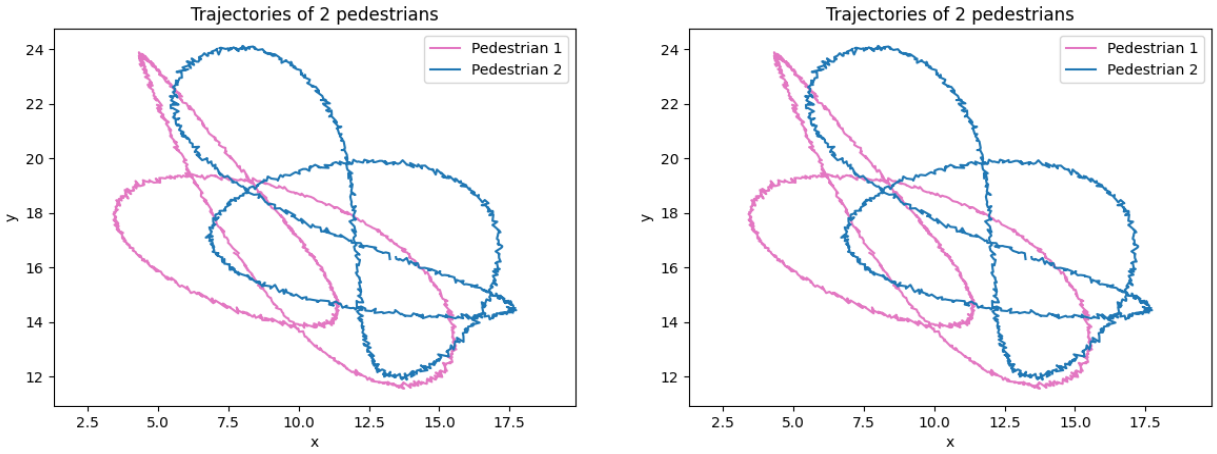
- **Original principal components**
  As shown in Fig.7(a), we can observe the original trajectories of the first 2 pedestrians. These trajectories appear disorganized, possibly due to the presence of noise in the data.
- **2 principal components**
  To address this, we applied PCA with 2 principal components to reconstruct the trajectories. The result is shown in Fig.7(b). These 2 components capture 84.92% of the energy in the data, with an energy loss of 15.08%. However, they do not capture the majority of the energy (>90%). Additionally, we notice that the reconstructed trajectories in Fig.7(b) are significantly distorted compared to the original one. This indicates that using only 2 components is insufficient to capture most of the energy in the data.



(a) (b)

Figure 7: The trajectories of 2 pedestrians [a] the original path [b] the reconstructions of the path with 2 principal components.

- **3&4 principal components**
  To determine the number of principal components needed to capture the majority of the energy, we implemented a conditional loop. And we have a dataset with 30 variables and 1000 records. Now we apply PCA to this dataset and analyze the energy loss as shown in Fig.9. The result indicated that at least 3 principal components are necessary. As shown in Fig.8(a) and Fig.8(b), the reconstructed trajectories use 3 and 4 principal components. They look the same and they both capture 99.71% of the energy in the data, with an energy loss of 0.29%. And we can observe that the shapes of the trajectories are preserved and resemble the original ones. Moreover, the trajectories appear smoother than the original due to the noise-filtering capability of PCA. Therefore, PCA effectively filters out the noise in this particular case, enhancing the quality of the reconstructed trajectories.



(a) (b)

Figure 8: The trajectories of 2 pedestrians [a] the reconstructions of the path with 3 principal components [b] the reconstructions of the path with 4 principal components.

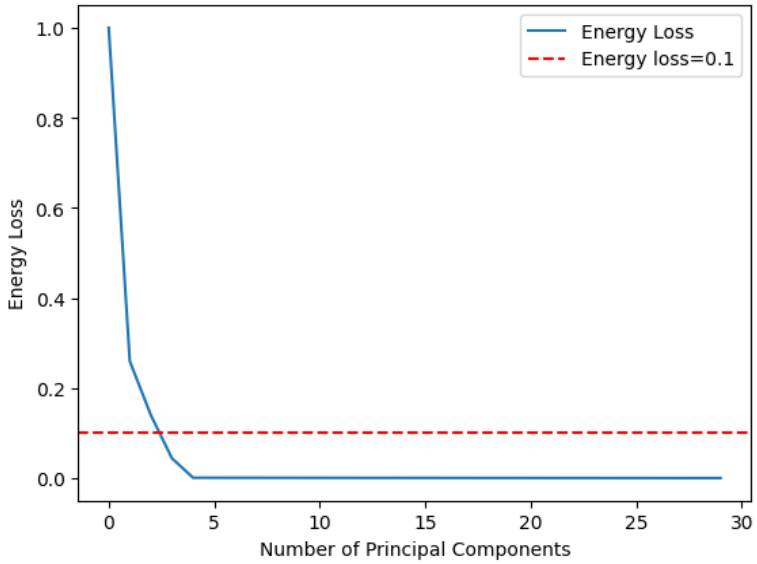


Figure 9: The energy loss with the different number of principal components.

Report on task 2, Diffusion Maps

Task description: We are expected to implement the Diffusion Map[1] algorithm ourselves. Therefore, three Python files and one notebook file were created to execute the various assignments.

- **dataset.py**
  From this file, we could get the required dataset used for different parts, specifically: periodic data and Swiss roll data. Then, visualize them into 2D or 3D versions.

- **diffusion_maps.py**
  Most utilities for diffusion maps algorithm are performed by this file. It consists of two functions, one is *diffusion_map_algorithm* enables the computation of eigenfunctions by following the guide provided in the worksheet. Another one is *diffusion_map_datafold_bonus* which utilizes the datafold library to calculate eigenfunctions.

- **diffusion_maps_plot.py**
  The primary purpose of this file is to generate plots of the eigenfunctions for different parts of the task. To facilitate this, the file includes a parameter called "part" that allows for switching between different sections to visualize the computation results as there are three parts and an additional bonus part in task2.

- **Task2_DiffusionMaps.ipynb**
  This file serves for task visualization purposes. By importing the three files mentioned earlier, along with essential libraries, and defining crucial hyperparameters like the number of samples and desired eigenfunction count, after executing the code, it allows us to conveniently observe and analyze the dataset and the corresponding plotted visualization result for specific requirements.

Part 1: Demonstrate the similarity of Diffusion Maps and Fourier analysis

In the first part, we need to work on periodic data set with N=1000 points (Fig.10). It is given by the following equations:

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

By applying the Diffusion Maps algorithm, the obtained five largest eigenfunctions ϕ_l associated with the largest eigenvalues λ_l are shown in Fig.11

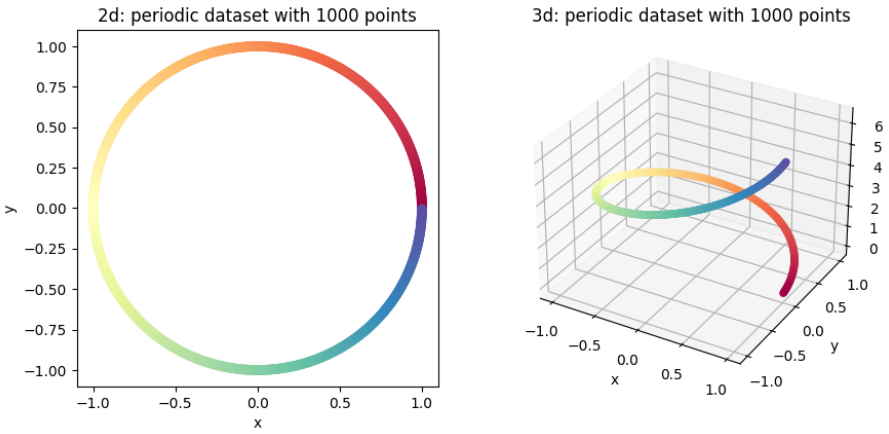


Figure 10: A periodic dataset

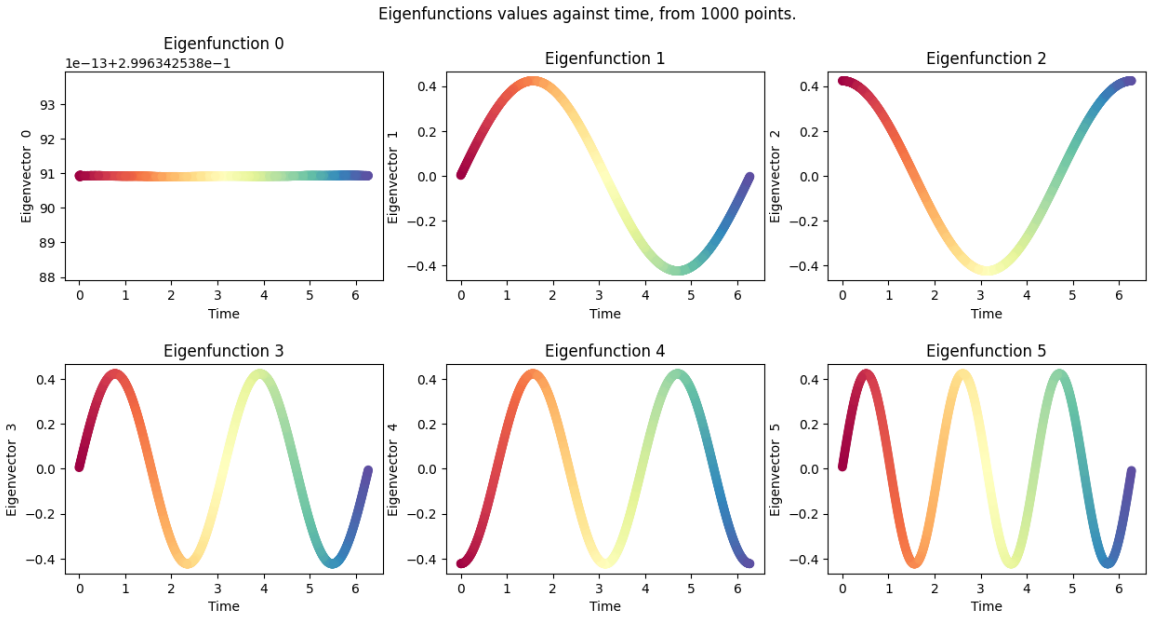


Figure 11: Visualization for values of eigenfunction ϕ_l against t_k

From Fig.11, upon observation, we can note that the largest eigenfunction ϕ_0 remains constant. Furthermore, as the frequency increases, all the eigenfunctions within the provided dataset exhibit sinusoidal behavior and undergo periodic transformations over time. These observations suggest the presence of a correlation between Diffusion Maps and Fourier Analysis. Both methods rely on measuring similarities or relationships between data points.

Part 2: Analyze on "Swiss roll" manifold

Task description: We need to obtain the first ten eigenfunctions of the Laplace Beltrami operator on the "Swiss Roll" manifold generated by using **sklearn** Python library. In this case, this dataset is created by 5000 points with 0.1 Gaussian noise. As shown in Fig.12 (Manifold: a lower-dimensional constrained "surface" upon which the data is embedded [2])

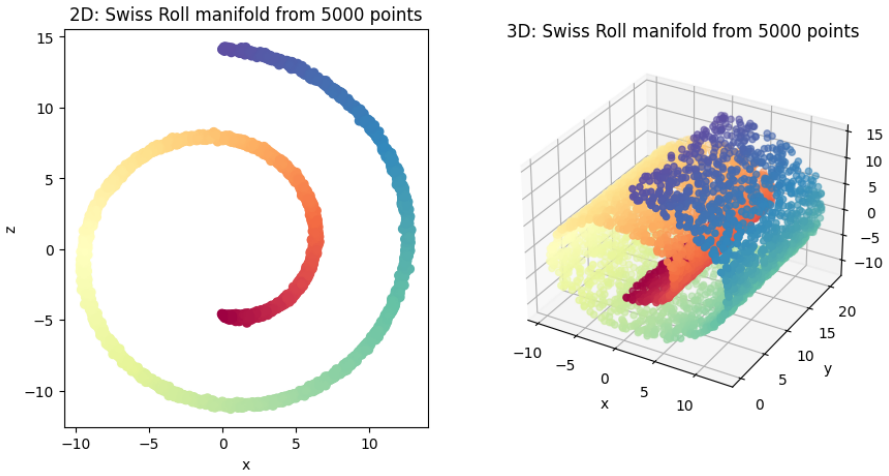


Figure 12: A Swiss Roll dataset with 5000 data points

By applying the Diffusion Maps algorithm, we obtained the approximations of the largest ten eigenfunctions, the plot for the first non-constant eigenfunction ϕ_1 against the other eigenfunctions is shown in Fig.13

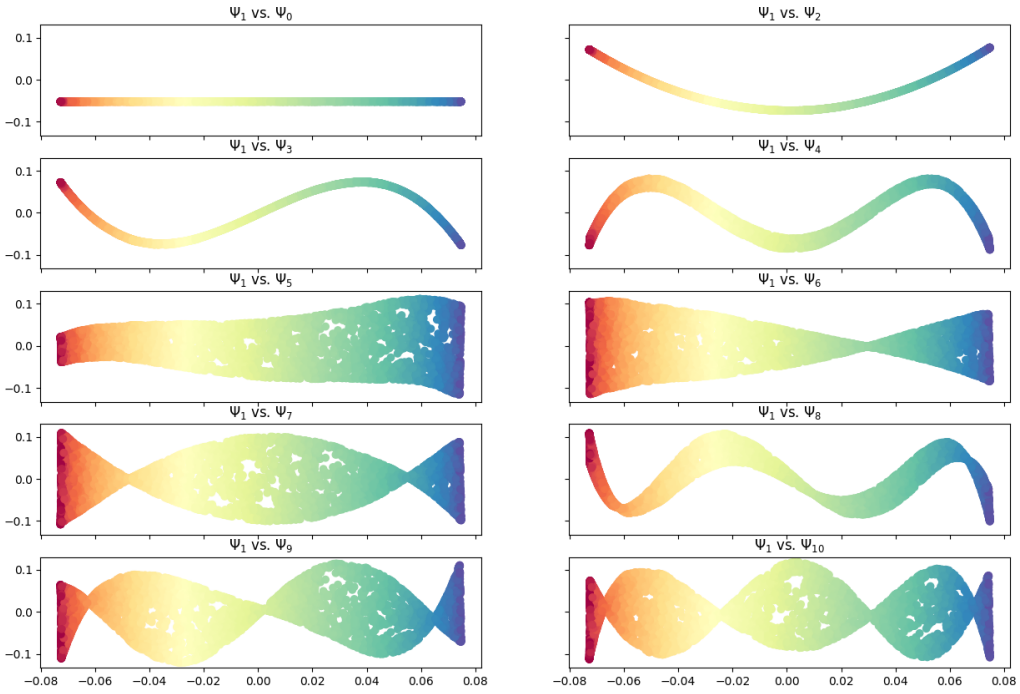


Figure 13: The first non-constant eigenfunction ϕ_1 against the other eigenfunctions with 5000 data points

From the above figure, we can find that when l=5, ϕ_l is no longer a function of ϕ_1. Since the associated eigenvalues λ_l indicate the importance of each dimension. Dimensionality reduction is achieved by retaining the k dimensions associated with the dominant eigenvectors.[2] As a comparison, now, we try to use the PCA algorithm to calculate the three principal components of the Swiss roll dataset. When we are choosing to keep three principal components to represent the dataset, the energy loss as expected is 0. This means the reconstruction dataset actually is the origin one. Compared to the PCA algorithm, Diffusion Maps have the advantage of better unfolding or capturing the underlying structure of complex and nonlinear datasets. While PCA is effective for linear data patterns, it may struggle to reveal the intricate relationships and nonlinear variations present in the data. The corresponding graph is shown in Fig.14.

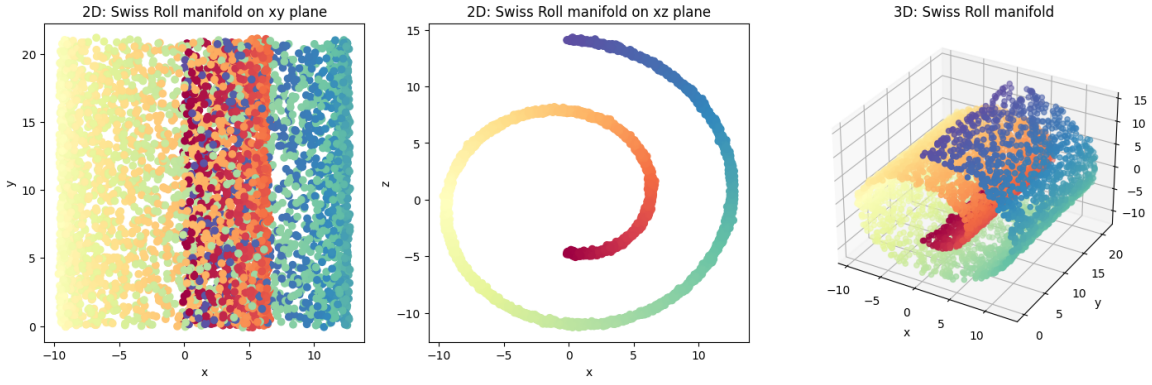


Figure 14: Reconstruction dataset with three principal components

When attempting to retain the top two principal components for the given dataset, it is worth noting that the PCA algorithm can capture the curvature to some extent. However, it may come at the cost of significant energy loss(30%). The result for dimensionality reduction is shown in Fig.15.In a word, PCA is suitable for linear data patterns, Diffusion Maps outperform PCA in unfolding complex and nonlinear datasets by preserving the intrinsic structure and capturing the intricate geometry present in the data.

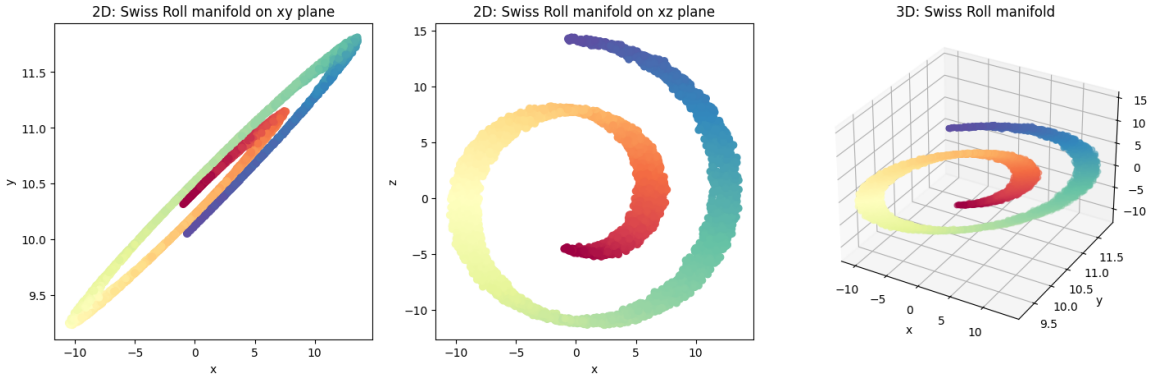


Figure 15: Reconstruction dataset with two principal components

What happens if only 1000-point data points are used? As illustrated in Fig.16, even when the dataset size is reduced to 1000, the shape of the Swiss roll can still be recognized. However, it is important to note that the eigenfunctions obtained from Diffusion Maps may exhibit increased noise compared to when the dataset contained 5000 points. Analogically, the same analyze for applying the PCA algorithm on the Swiss roll data set with 1000 data points. The result is shown in Fig.18 and Fig.19

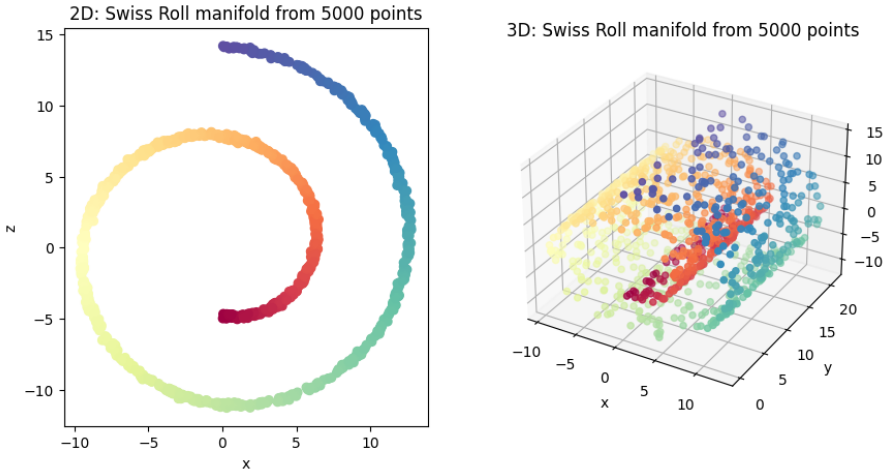


Figure 16: A Swiss Roll dataset with 1000 data points

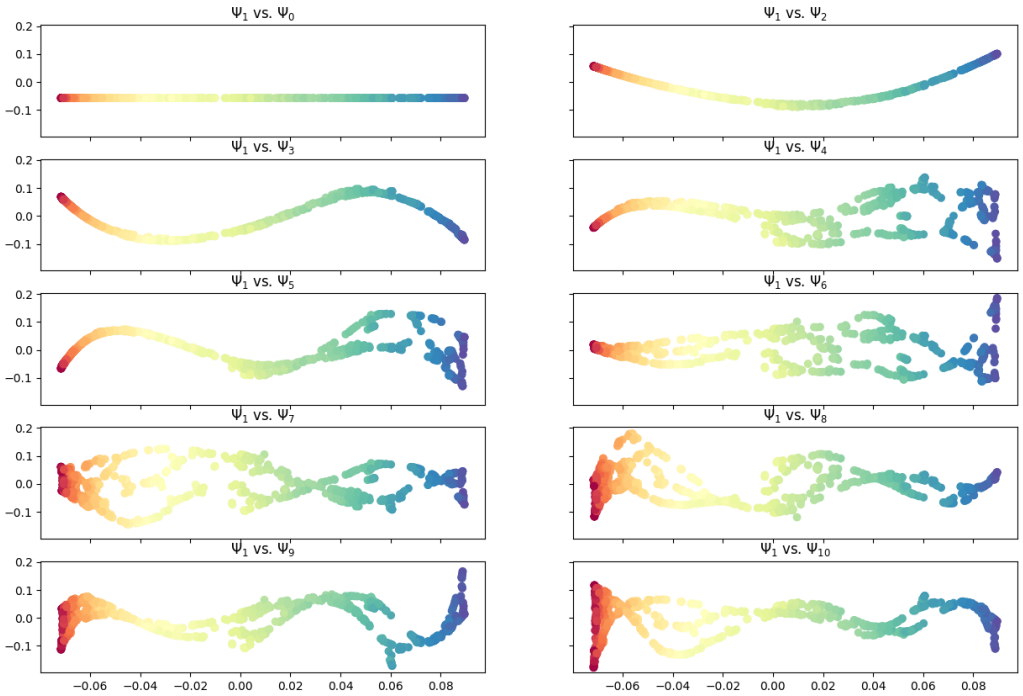


Figure 17: The first non-constant eigenfunction ϕ_1 against the other eigenfunctions with 1000 data points

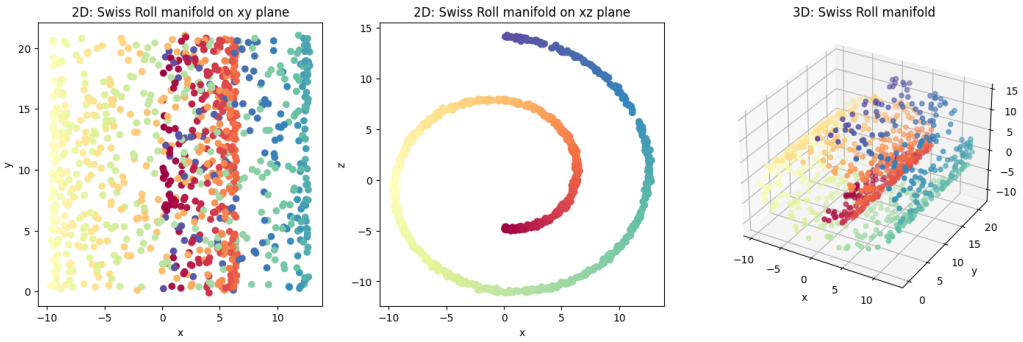


Figure 18: Reconstruction dataset with three principal components with 1000 data points

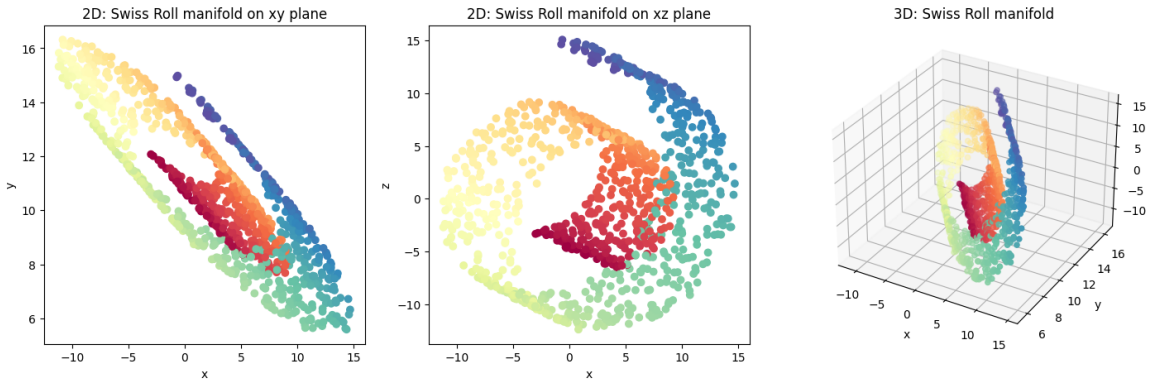


Figure 19: Reconstruction dataset with two principal components with 1000 data points

Part 3: Work with trajectory data

In this sub-task, we perform the Diffusion Maps on **data_DMAP_PCA_vadere.text** in order to analyze the trajectory data. First, we take a glance at what the trajectory path looks like. Illustrated in Fig.20. As the Diffusion Maps don't have exactly the same energy interpretation of the eigenvalues(singular values) as PCA (Note form worksheet), so we need to find another data representation accurately by proper eigenfunctions combination. Here proper data representation means there are no "intersections" of the unfolded data in the latent space. With the same analysis as the previous part, we choose the largest ten eigenfunctions ϕ_1 to unfold the data and capture the relevant information. Here we present the visualization results as shown in Fig.21. Each subgraph shows the possibility of no intersections. In such 2D space, we could easily find that the subspace spanned by ϕ_1 and ϕ_2, so it's a good combination of two eigenfunctions.

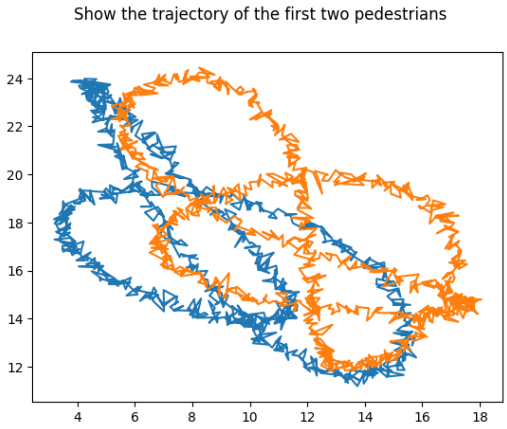


Figure 20: The plot for the trajectory of two pedestrians

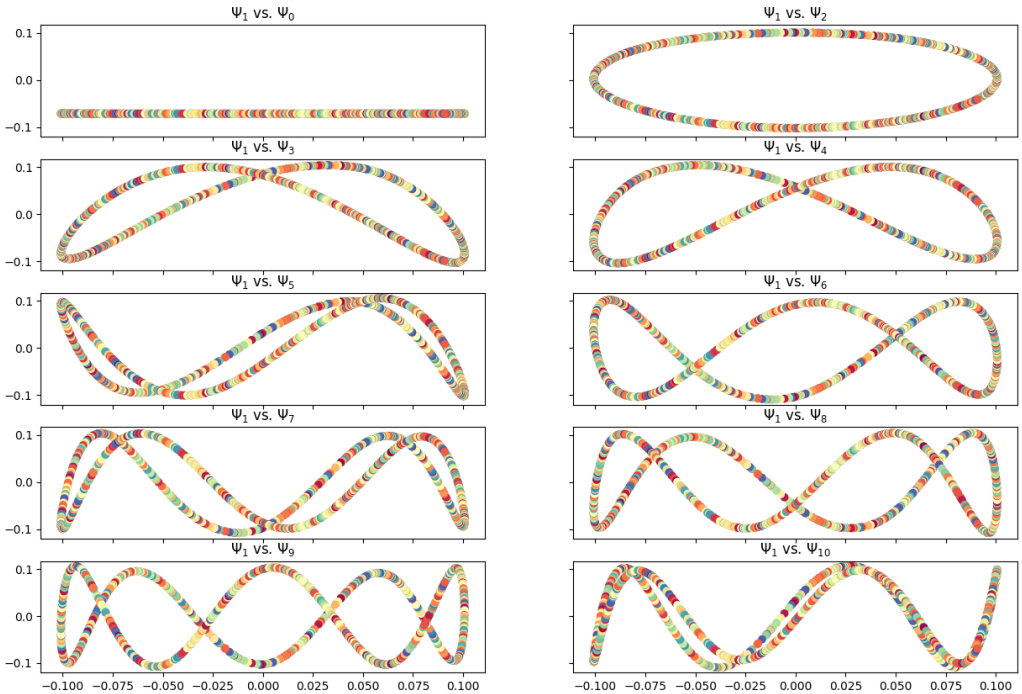


Figure 21: The plot for the eigenfunction ϕ_1 against the other eigenfunctions in 2D

Part 4: Bonus part, compute eigenvectors of Swiss roll data set with datafold

In this section, we revisit the Swiss roll dataset and utilize the "datafold" software to apply the same process as demonstrated in Part 2. The "datafold" algorithm proves to be superior to our previous approach. It offers code that is both concise and intuitive, while also being optimized for improved performance and efficiency. The result is shown in Fig.22.

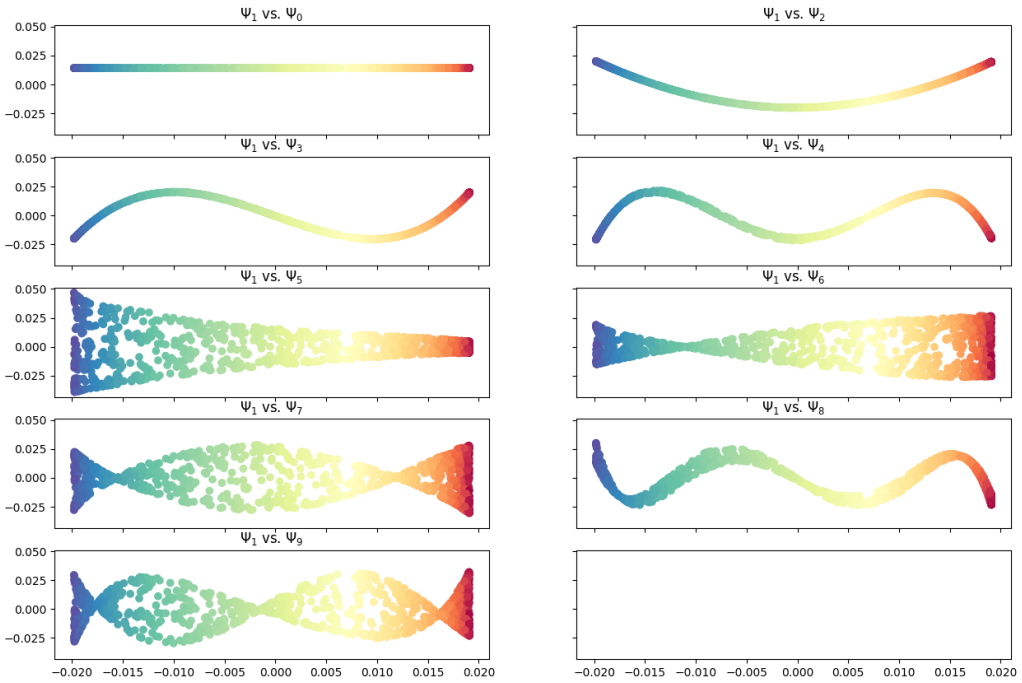


Figure 22: The first non-constant eigenfunction ϕ_1 against the other eigenfunctions by datafold

Verbose discussion of the result on task 2

Finishing the implementation of task2, almost took us 4 days, one day for the knowledge preparation related to Diffusion Maps and Datafold, Two days to implement the methods, and one day to fix bugs. This method is a new field for us, before, we just realized some basic knowledge about PCA, VAE, Matrix Factorization, etc from the lectures. Many thanks to the article [2], it really provided a friendly version for beginners like us to study with explaining profound theories in simple language.
Usually, the datasets from our real world are complex and within high-dimensional space. For people to understand and interpret the dataset more easily and intuitively, the dimensionality reduction method and visualize them in 3D or 2D are essential. Diffusion Maps Algorithm is such a good reduction method that focuses on discovering the underlying manifold with non-linear datasets. Unlike PCA (suitable for linear data), Diffusion Map Algorithm has the ability to capture the global structure for the given dataset by integrating local similarities at different scales. It measures the local similarities between two points by applying a random walk on the dataset using the connectivity property. We first define a diffusion kernel to express the connectivity within a certain neighborhood, it enables us to build a diffusion matrix P, and each entry provides the probability from i jump to j in a single timestep, With increasing the time steps, we could observe the dataset at different scales by calculating P^t, which it called diffusion process, by doing so, the underlying geometric structure of the given dataset revels. The eigenfunctions and their associated eigenvalues of P represent the dominant patterns and variations within the data. The first eigenfunction corresponds to the dominant global structure, followed by successive eigenfunctions representing increasingly finer details.
We visualize all the results of the given data after dimension reduction to check the accuracy of our model conveniently. Like checking the preservation of the local and global structure. For the periodic dataset, the eigenfunctions' value should keep the same at the start and final timesteps. In the Swiss roll case, we expected the reduced data should present an unfolded dataset in a 2D subspace.

Report on task 3, Variational AutoEncoder

1. Introduction

Motivation. Autoencoders are unsupervised learning models that aim to learn a compact representation of the input data. They consist of an encoder network that maps the input data to a latent space and a decoder network that reconstructs the input data from the latent representation. However, traditional autoencoders have certain limitations, such as the lack of control over the latent space and the generation of unrealistic samples. **Variational Autoencoders (VAEs)**[3] were introduced to address these flaws and offer a more powerful generative modeling framework. The key idea behind VAEs is to introduce probabilistic modeling into the latent space of autoencoders. Instead of directly mapping the input data to a fixed latent representation, VAEs learn the parameters of a probability distribution in the latent space. This allows for a more flexible and expressive representation of the input data.

Model Architecture. VAE model contain two parts: An encoder $q_\phi(z|x)$ and an decoder $p_\theta(x|z)$, which can be modeled by neural networks. The details are shown in Fig.23.

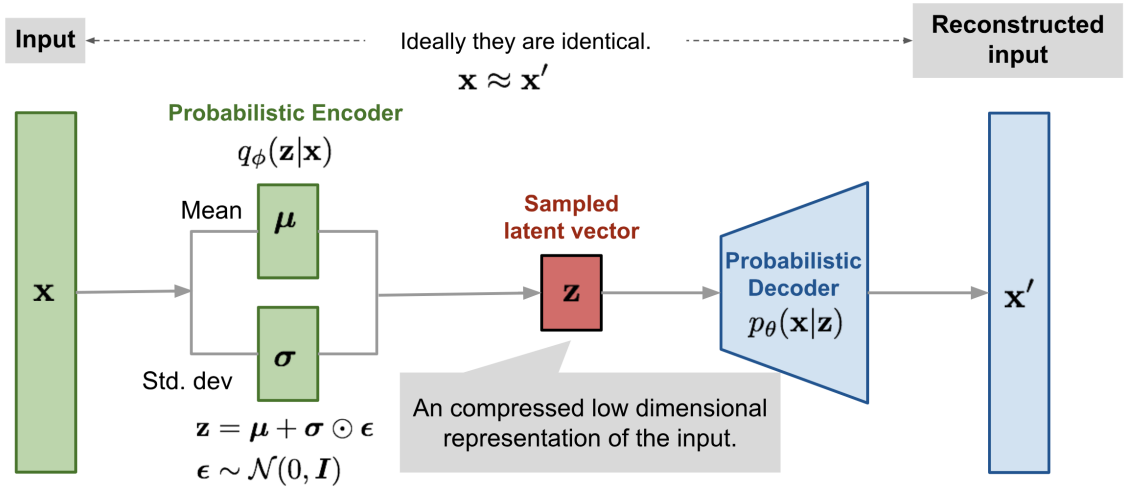


Figure 23: An illustration of the Variational AutoEncoder model.[4]

Model Training. During training, VAEs optimize two objectives simultaneously. The first objective is the reconstruction loss, which measures the difference between the input data and the reconstruction generated by the decoder network. The second objective is the Kullback-Leibler (KL) divergence, which encourages the learned latent distribution to resemble a predefined prior distribution (usually a multivariate Gaussian). The KL divergence regularizes the latent space and helps prevent overfitting. Specifically, the loss of VAE is:

$$\begin{aligned} L(\theta, \phi) &= -\log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \\ &= -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) \\ \theta^*, \phi^* &= \arg\min_{\theta, \phi} L(\theta, \phi) \end{aligned}$$

Mathematical Deduction

Our goal is to derive $p_\theta(x)$, which is hard to be solved directly. An idea is to match it to another distribution $p(z)$, where z can be easily used to control the generation process. By Bayes formula, we can find a easy association between x and z:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}$$

Though the math is simple and beautiful, this formula is usually intractable since it is computation expensive to sample lots of z from $p(z)$. Researches find that we can derive an low bound of $p(x)$ and optimize the low bound, which is called **Evidence Lower BOund(ELBO)**. Now we show how to derive it.

$$\begin{aligned}\log p_\theta(x) &= \log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)} \\ &= \log \frac{p_\theta(x|z)p_\theta(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)} \\ &= \log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p_\theta(z)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)}\end{aligned}$$

Thus,

$$\begin{aligned}\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(x) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(x|z) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log \frac{q_\phi(z|x)}{p_\theta(z)} + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(x|z) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})\end{aligned}$$

Then, we can get this equation:

$$\log p_\theta(x) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(x|z) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))$$

The LHS of the equation is exactly what we want to optimize: $\log p_\theta(x)$ is the log-likelihood of the data we need to maximize, and $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})$ represents the difference between the real and the estimated posterior, which we need to minimize. We can also derive the inequality:

$$\log p_\theta(x) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(x|z) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) = L_{ELBO}$$

, since any KL-divergence is greater or equal than 0. This inequality reveals that L_{ELBO} is a lower bound of the data distribution $p_\theta(x)$. Thus we can use L_{ELBO} to optimize generation of data.

KL-divergence of 2 Gaussian Distribution. The KL-divergence of 2 Gaussian Distribution $N(\mu_1, \sigma_1)$, $N(\mu_2, \sigma_2)$ is:

$$D_{\mathrm{KL}}(N(\mu_1, \sigma_1)\|N(\mu_2, \sigma_2)) = \log \frac{\sigma_2}{\sigma_1} - \frac{1}{2} + \frac{\sigma_1^2}{2\sigma_2^2} + \frac{(\mu_1 - \mu_2)^2}{2\sigma_2^2}$$

For VAE, we usually use $N(0, 1)$ as the prior $p_\theta(x)$. With the equation showed above, the KL-divergence of $N(\mu, \sigma)$ and $N(0, 1)$ is:

$$D_{\mathrm{KL}}(N(\mu, \sigma)\|N(0, 1)) = -\log \sigma - \frac{1}{2} + \frac{\sigma^2}{2} + \frac{\mu^2}{2}$$

Empirical Loss. Practically, we use the loss:

$$L = \sum_{X\in B} [\|X - \hat{X}\|_2^2 + (-\log \sigma - \frac{1}{2} + \frac{\sigma^2}{2} + \frac{\mu^2}{2})]$$

, where B the set of batches of data, $\hat{X}$ is the output of VAE, σ and μ is approximated in the encoder of VAE.

Advantages of VAE. By incorporating the probabilistic nature of the latent space, VAEs offer several advantages over traditional autoencoders.

1) Firstly, they enable the generation of new data samples by sampling from the learned latent space distribution. This allows for controlled and structured generation of data.

2) Secondly, VAEs provide a continuous and structured latent space, allowing for smooth interpolation and exploration of the data manifold.

3) Lastly, the regularizing effect of the KL divergence helps prevent overfitting and improves the generalization capability of the model.

2. Answering Question 1 and 2

We have conduct some experiments to show the properties of Variational AutoEncoder, and answer the questions provided in the exercise sheet.

Activation Functions. Mean parameter of the approximate posterior and the likelihood: Identity activation function. Allows the network to output arbitrary real values without transformation. Standard deviation parameter of the approximate posterior: exponential activation function. Maps the input to a positive range. Ensures valid and meaningful values for the variance.

Reconstruction-Generation Trade Off. If you obtain good reconstructed digits but bad generated digits in a variational autoencoder (VAE), it could indicate that the VAE is successfully learning to capture the input data distribution but is struggling to effectively generate new samples from the learned latent space. Here are a few possible reasons for this issue:

1) Latent space limitations: The latent space representation in the VAE might not be expressive enough to capture the full complexity and diversity of the target digit distribution. The encoder-decoder architecture and the dimensionality of the latent space might need to be adjusted to allow for more flexibility in generating diverse samples.

2) Inadequate prior distribution: The choice of the prior distribution in the VAE, typically a multivariate Gaussian distribution, may not be well-suited for the target digit distribution. Exploring different prior distributions, such as more flexible or structured distributions, could lead to better generation performance.

3) Insufficient training: It's possible that the VAE hasn't been trained for a sufficient number of epochs or with an appropriate learning rate. Training a VAE can be challenging, and it might require longer training or hyperparameter tuning to achieve better generation quality.

4) Mode collapse: Mode collapse occurs when the VAE fails to capture the full diversity of the data distribution and instead focuses on a few dominant modes. As a result, the generated digits may lack diversity and appear similar or repetitive. Techniques like incorporating regularization strategies (e.g., KL divergence annealing) or using more advanced architectures (e.g., Wasserstein VAEs or VAE-GAN hybrids) can help alleviate mode collapse.

5) Data limitations: If the training dataset lacks diversity or is skewed towards certain types of digits, it can limit the VAE's ability to generate realistic and diverse samples. Collecting or augmenting the training data with a wider range of digit variations could help improve generation quality.

6) Hyperparameter tuning: The VAE hyperparameters, such as the learning rate, batch size, network architecture, and regularization terms, can significantly impact the model's performance. Experimenting with different hyperparameter settings might lead to better generation results.

3. Experiment

Experimental Settings

1) **Posterior(Encoder)** $q_\phi(z|x)$ **and Likelyhood(Decoder)** $p_\theta(x|z)$: Both are 2 fully connected layers with 256 units, using $ReLU$ activate functions. (2-256-256-2)

2) **Prior** $p(z)$: A Gaussian Distribution $N(0, 1)$

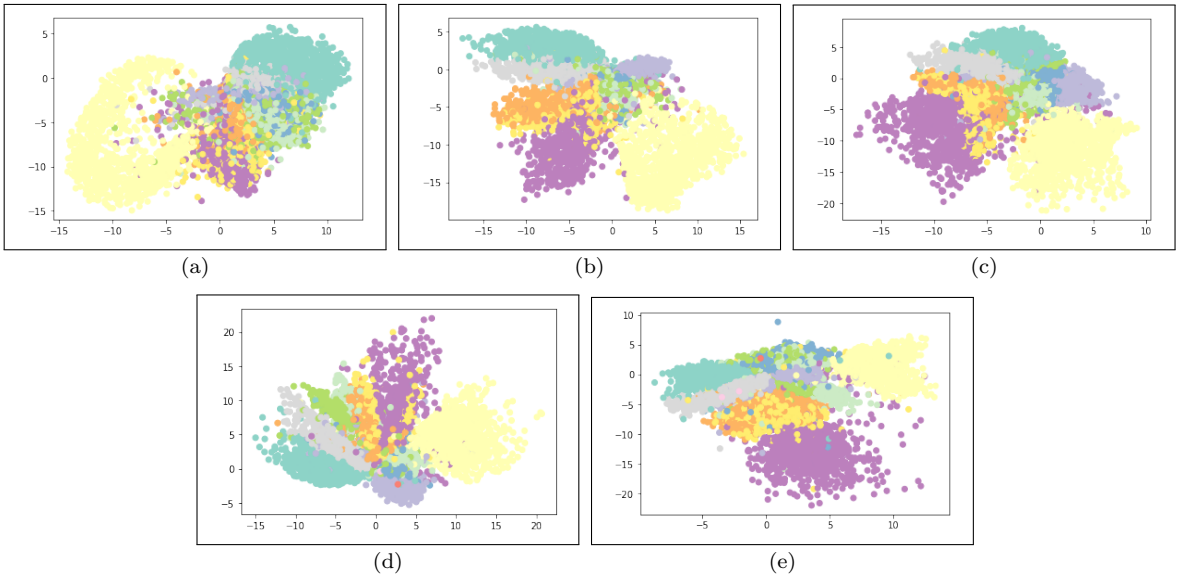
3) **Learning Rate**: 0.001

4) **Optimizer**: Adam

5) **Batch Size**: 128

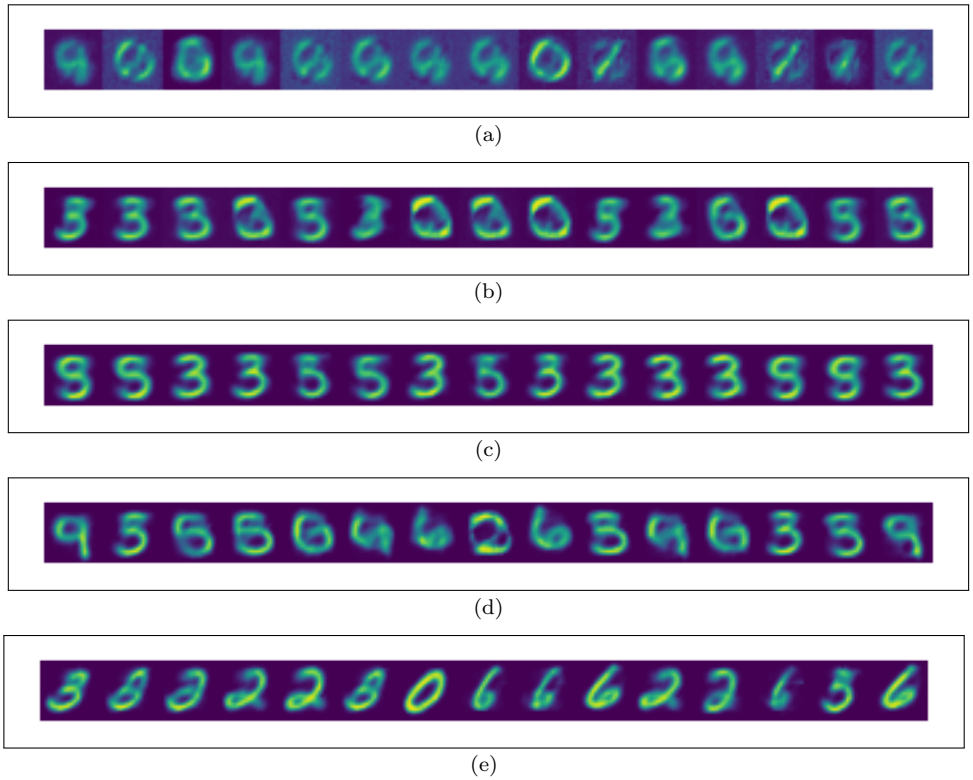
Visualization. As instructed by the exercise sheet, we will visualize some latent representations, test-set reconstructions, and generated data after the 1st epoch, the 5th epoch, the 25th epoch, the 50th epoch, and after the optimisation converged. Latent representations, test-set reconstructions, and generated data are shown in Fig.24, Fig.25, and Fig.26 respectively. As we can see from results, during training, the latent space is having

a more clear boundary; the reconstruction is becoming more precise; and the generation data is becoming more clear.



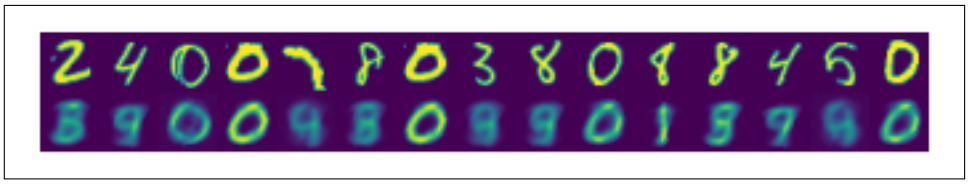
(a) (b) (c) (d) (e)

Figure 24: Visualization for latent space of testing data. 24(a), 24(b), 24(c), 24(d), and 24(e) are for the 1st epoch, the 5th epoch, the 25th epoch, the 50th epoch, and after the optimisation converged respectively.

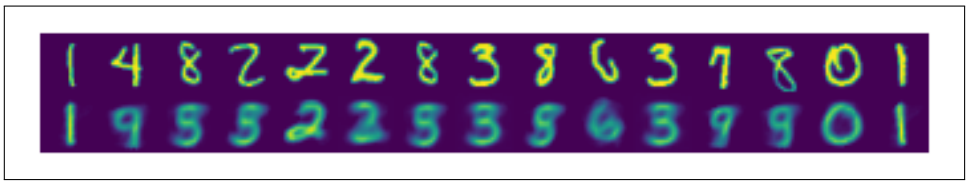


(a) (b) (c) (d) (e)

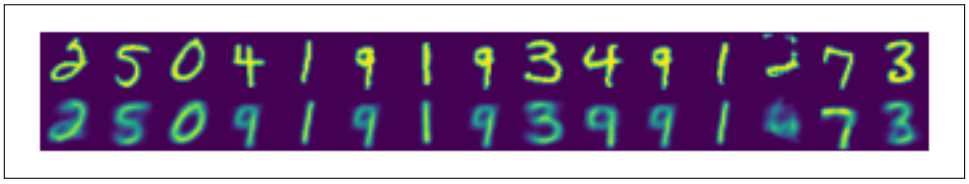
Figure 25: Visualization for new data generation. 25(a), 25(b), 25(c), 25(d), and 25(e) are for the 1st epoch, the 5th epoch, the 25th epoch, the 50th epoch, and after the optimisation converged respectively.



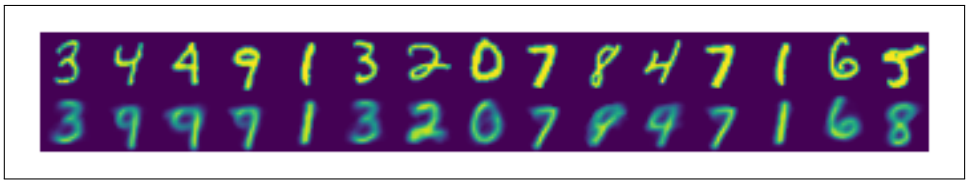
(a)



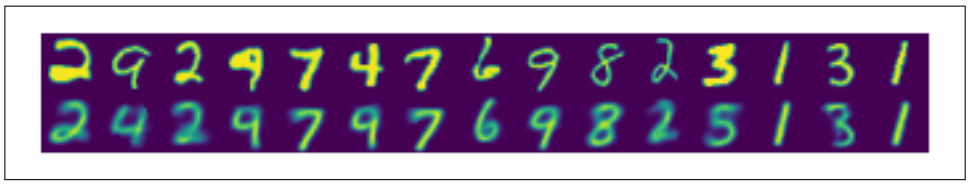
(b)



(c)



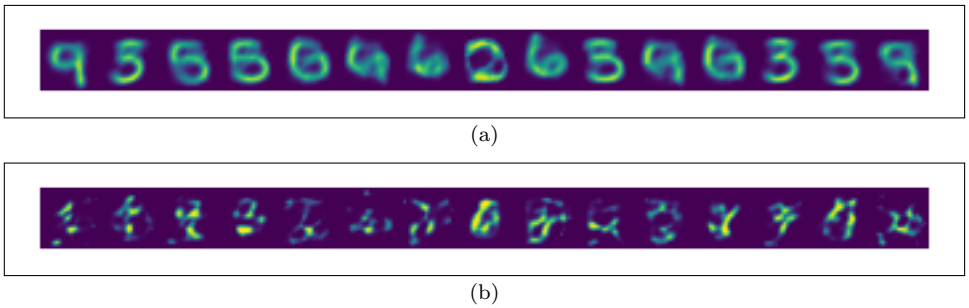
(d)



(e)

Figure 26: Visualization for testing data reconstruction. The upper one is the original data and the lower one is the reconstruction data in each figure. 26(a), 26(b), 26(c), 26(d), and 26(e) are for the 1st epoch, the 5th epoch, the 25th epoch, the 50th epoch, and after the optimisation converged respectively.

VAE Using a 32-Dimensional Latent Space. We have constructed another VAE model using 32-dimensional latent space. To show its properties with expanding latent dimensions, we compare its results with those in 2-dimension one. Specifically, we compare their generated data and loss curves. The comparison for generated data is shown in Fig.27, and the comparison for loss curves is shown in Fig.28. The result of VAE using 32-d latent space is totally worse than that of 2d-space, and it nearly can't reconstruct the data. It is probably caused by **Curse of Dimensionality**. As the latent dimension increases, the volume of the space grows exponentially. This can cause issues with the density estimation performed by the VAE. The model might struggle to capture the underlying data distribution accurately due to the sparsity of samples in high-dimensional spaces.

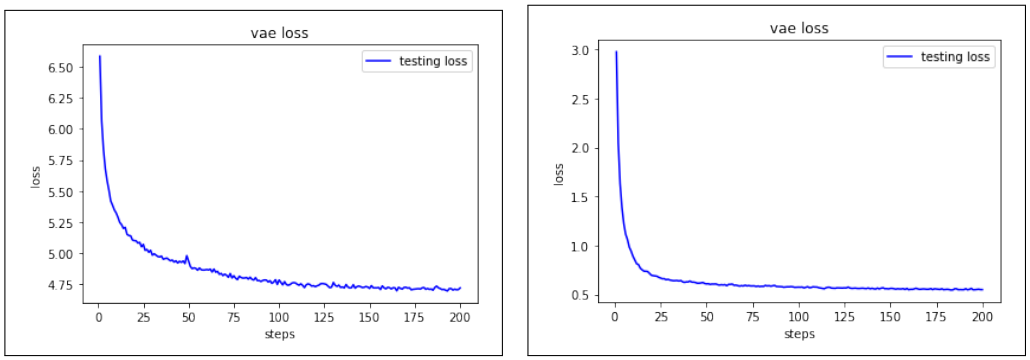


(a)

(b)

Figure 27: New generated data. 27(a) and 27(b) are generated by using Gaussian noise.

Loss Curves



(a) (b)

Figure 28: Loss curves. The x-axis and y-axis represent the number of epochs and loss values. 28(a) and 28(b) are loss curves of VAE with 2-d and 3-d latent space respectively.

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

To properly model and train an autoencoder for an evacuation plan at TU Munich MI department, based on the given datasets we first need to apply some preprocessing steps.

To be able to work with normalized data we want the training samples' positional information we want their x and y positions to be in the range of $[-1, 1]$. Therefore we define the minimum possible coordinate value $c_{min} = 0$ and $c_{max} = 200$ for x and y coordinates respectively and perform the normalization step by

$$c_{normalized} = \frac{c - c_{min}}{c_{max} - c_{min}} * 2 - 1$$

with c being a single x or y coordinate value. This transforms the values from $c \in [0, 200]$ to $c_{normalized} \in [-1, 1]$

To be able to work with generated data we also define the reverse process mapping generated data coordinates $c_{normalized}$ back to their original range.

$$c = (c_{max} - c_{min}) * \frac{c + 1}{2} + c_{min}$$

Model Based on the normalized data and the recommended model parameters and guidelines we define our encoder model with two linear layers combined with respective ReLU activation functions to add nonlinearity to the model and two linear layers to incorporate the learning of mean and standard deviation of our training samples. For the decoder we use three linear layers, complemented again with ReLU activation functions.

Endoder Trivially the encoder starts with a linear layer with an input dimension of two, for the representation of the coordinate space. From there on we choose the first Linear layer to go to a 128 feature space and 2 dimensional latent space.

Decoder The decoder hast 3 linear layers, two of them with feature space 128 and the third one mapping to the output of the Autoencoder with of course again 2 dimensions being the coordinate space.

To train the model, the following hyperparameters are used:

```
1 hidden_dims: 128
2 latent_dims: 2
3 learning rate schedule: lr = [0.01, 0.001, 0.0001, 0.00001]
4 epochs = 200
5 lambda = 100
```

The result of training the model is illustrated in Fig.29.To achieve better performance in the training process, here we choose a learning rate schedule that is a linear schedule with an lr drop instead of setting a fixed learning rate. In addition, we also defined a variable called *lambda* which enables the model to get a trade-off between reconstruction and generation.

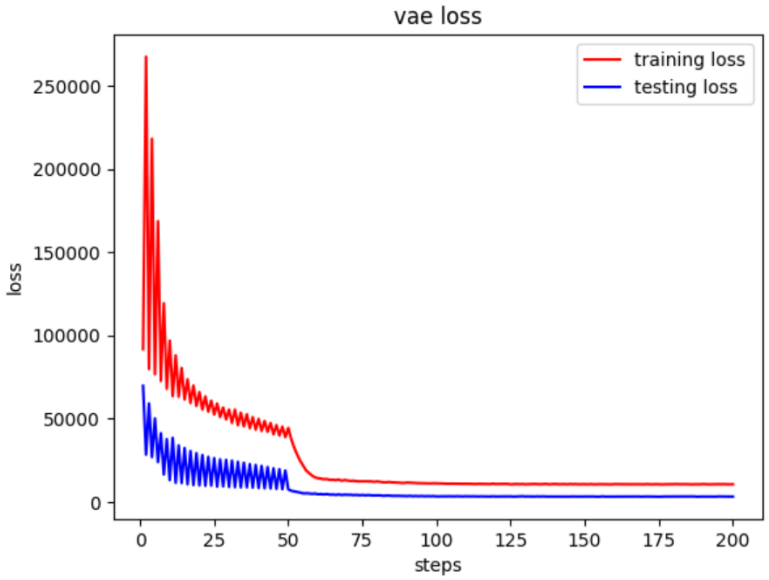


Figure 29: Training (red) and test(blue) loss curves

Generated Data Using the pre-trained model we generated 1000 new samples and compared the results to the original training and test data, visualized in Fig.30. It can be seen that, even though the decoder output does not very precisely model the complete input distribution in all its details we can see recreated tendencies in critical parts which allow us to analyze people densities in this area. Gradually generating data until the maximum threshold of 100 people in the critical area is reached, yields that for around 750 generated samples this critical population threshold is reached.

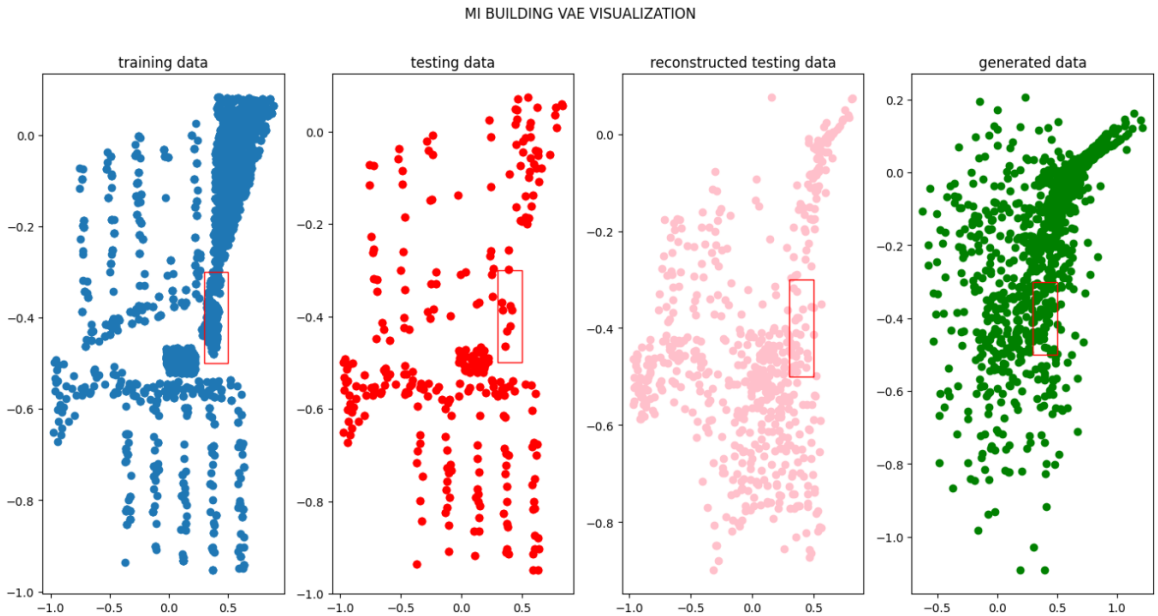


Figure 30: Visualization of single datasets with critical location marked through red box

From task 4, we could realize that Variational Autoencoder goes beyond being a technique solely used for dimension reduction and can also be effectively employed to learn and generate samples from intricate distributions.

References

[1] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006. Special Issue: Diffusion Maps and Wavelets.

[2] Jacqueline Delaporte, Ben M. Herbst, Willy A. Hereman, and Van der Walt Stéfan. An introduction to diffusion maps. 2008.

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

[4] Lilian Weng. From autoencoder to beta-vae. *lilianweng.github.io*, 2018.