Introduction to Intelligent Systems
Lab Session 1

Group 17
Tobias Pucher (s5751659) & Thanakit Chaichanawong (s5752094)

September 20, 2023

Assignment 1

1. Introduction

When thinking about processing data for machine learning, typically there will be a massive amounts of data that has to be processed. In a computer, this will be seen as variable or dimension, and the number of dimensions is usually quite large. On the other hand, with large amounts of data, it typically require s considerable amount of storage and computation complexity. Dimensionality reduction is one of the method that is used for reducing the amount of data stored with some defined amount of information loss as is further explained in Simplilearn [2]. It is also one of the ways to mitigate over-fitting by reducing the model complexity and noise in the data as described in Pramoditha [1].

This experiment aims to create a PCA function, which is one of the methods of dimensionality-reduction with mathematics and matrix operation. The input data will be the 1440 pictures of 20 distinct objects in form of flatten array that can be reconstructed to be a matrix. Each 20 objects have 72 different pictures with the size of 32x32 that each picture is 5 degree difference in rotation from each other. With these data set, there will be some un-important data points that can be removed with insignificant difference from the original data. In this experiment, a PCA function will be implemented and experimented with, and also the reconstruction accuracy will be observed by the fraction of variance.

2. Methods

As far as we understood the concept, PCA basically works out the contributions of each variable in the dataset to the overall variance of the data set, picks the best linear combinations (principal components) of all variables, and projects the data into that lower dimensional space according. This is done to maximize the information in the new data set when reducing its number of variables (dimensions). Therefore the already mentioned principal components are calculated. Principal components are new variables constructed from a mixture of all original variables, such that they represent a direction in the data that maximizes variance. Therefore the larger the variance the larger the information along that direction / axis. For a n-dimensional data set there are exactly n-principal components. PCAs goal is to find these and rank them according to their importance and pick the top d components. The first calculation step is centralizing the data set and calculating the centralized data sets covariance matrix. From this eigenvalue decomposition is done to get the eigenvalues and eigenvectors of the covariance matrix. In this decomposition all n principal components are found, with their corresponding eigenvalue. The eig-vector gives the direction (new axis) and its eigenvalue the amount of variance in that direction. For each axis its contribution can

be calculated by diving by the sum of all eigenvalues. By ranking these eigenvectors based on their values the best d eigenvectors are picked. The sum of these divided by the sum of eigenvalues is p_d, the fraction of total variance. This gives an indication of information that will be retained from the original data set. The first principal component accounts for the maximal amount of variance, the second one accounts for the second most amount of variance. Also each principal component is uncorrelated to the others (orthogonal). The last step is to project the data set onto U_d containing the eigenvectors. In this projection the eigenvectors (axis) become the basisvectors of the new data set. This reduced data set has therefore less dimension $d < n$ but in such a way that the most possible amount of information is retained.

Further implementation details are explained further down below.

2.1. PCA Implementation

To solve the problem of the PCA algorithm, we defined a method that takes the data set Z and targeted dimensionality d. The algorithm should then perform PCA on the data set and return a tuple with two elements: (Z_d, p_d) which represent the reduced data set with dimension d as well as the fraction of total variance p_d for given d. To speed up the development process we used the popular python library "numpy" for the various matrix calculations. The different steps of our PCA implementation will be explained in the following subsections.

2.2. Centralize the data set

In the following code-snipped Listing 1 the method definition and the first of the PCA algorithm is shown. This step prepares the data set by centralizing it. This is done by shifting the data set by the mean values. This is done using the column means (i.e for each variable) of the data set individually. The expression `mu = X.mean(axis=0)` creates a vector `mu` which includes the means μ for each variable (column). We then subtract this vector from the dataset `X`. Doing so subtracts each columns μ from the corresponding column in `X` for every row (measurement). The centralized data is stored in `Z`. The number of variables n, the dimension, of the data set is then calculated by the length of the first measurement. For this we assume that all measurements in the data set share the same number of variables. The dimensions of the matrix X and Z with m measurements and n variables are therefore: $X, Z \in \mathbb{R}^{m \times n}$.

Listing 1: *centralizing the dataset*

```
def PCA(X, d):
  mu = X.mean(axis=0)
  Z = ( X - mu )
  n = len(Z[0])

  l.info(f"# measurements: {len(Z)}")
  l.info(f"# variables: n={n}")

  if d >= n:
    l.error(f"invalid parameter: cannot reduce from {n} to {d}")
    return None
```

Our algorithm then performs some logging steps using pythons "logging" library as well as necessary input checks if a reduction is possible for given n and d.

2.3. PRINCIPAL COMPONENTS U AND EIGEN-VALUES D

To compute the principal components and eigenvalues the first step is to compute the covariance matrix of the centralized data set. Doing so enables us to find the eigenvectors ("optimal lines for projection") and eigenvalues ("importance of each component").

For the purpose of calculating the unbiased covariance matrix we utilized a method provided by numpy as shown in Listing 2. The parameter `rowvar=False` indicates that variables are contained in the columns. The result `covariance_matrix` is the matrix $C \in \mathbb{R}^{n \times n}$ containing the covariance between each of the variables on the given sample data, such that $C_{ij} = Cov(V_i, V_j) \forall i, j \in \mathbb{N} < n$. The `cov` function also bias-corrects each element. The output of this function call should therefore be equivalent to the math expression $C = \frac{1}{M-1} \cdot Z^T \cdot Z$. In hindsight, this equation could also have been implemented directly by using numpys dot function and then an per element division by $(M-1)$.

Listing 2: *covariance matrix and eigenvalue decompositon*

```
covariance_matrix = np.cov(Z, rowvar=False)
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
```

Furthermore in Listing 2 the eigenvectors and eigenvalues are then calculated using the method `np.linalg.eig(...)`. The returned eigenvectors each represent a principal component. To pick the best ones (with the highest "scaling") they are ranked by their corresponding eigenvalue. Higher eigenvalue results in a higher scaling factor when multiplied with a vector.

To rank the eigenvectors and associated eigenvalues based on the eigenvalues a sorted list of n tuples is created with elements $(v_i, u_i) : v_i \in \mathbb{R}, u_i \in \mathbb{R}^n$ as seen in Listing 3 . From there the first (most important) d tuples are picked.

Listing 3: *preparing the eigenvectors/values to be principal components*

```
tuples = sorted( (
  (eigenvalues[i], eigenvectors[i]) for i in range(len(eigenvalues))
  ), key= lambda x: x[0], reverse=True );
```

Next as displayed in Listing 4 a list of variance contributions is created for later use in calculating the fraction of total variance p_d. More importantly a matrix $U \in \mathbb{R}^{n \times n}$ is created with the n eigenvectors in the column and then taking the first d rows of U. This is the final U_d needed for projection in the next step.

Listing 4: *selecting the principal components U_d*

```
variance_contributions = [tup[0] for tup in tuples]
U = np.transpose([tup[1] for tup in tuples]) # columns contain eigen-
   vectors
U_d = U[:d]
```

The matrix U_d is of size $n \times d$.

2.4. CALCULATING INFORMATION RETAINED

The information retention (fraction of total variance) p_d is calculated in Listing 5 for given d by calculating $p_d = \frac{\sum_0^d V_d}{\sum V}$ with $V \ldots$ descending Eigenvalues. In the case of taking all eigenvectors would give $p_d = 1$ but of course without any reduction of the data. The factor p_d is the amount of variability retained in the reduced data when reduced in the following reduction step. A factor of 0.9 means 90% of variability is preserved.

Listing 5: *calculating fraction of total variance*

```
p_d = np.sum(variance_contributions[:d]) / np.sum(
    variance_contributions)
```

2.5. REDUCE THE DIMENSIONALITY OF THE DATA SET

The final step of reducing the dimensionality of the data set to d dimensions is shown in Listing 6 . This is done by multiplying the principal components with the centralized data set. This results in a projection of the data points to the reduced dimension d. The order of the dot product as well as the correct transposition of matrices ([m x n] dot [n x d] = [m x d]) is important to ensure that $Z_d \in \mathbb{R}^{m \times d}$. The resulting matrix has the same amount of measurements (rows) but with a reduced amount of columns (variables) with $Z_d = Z \cdot (U_d)^T$.

Listing 6: *projection to lower dimension with selected principal components*

```
Z_d = np.dot(Z, np.transpose(U_d))
return (Z_d, p_d)
```

2.6. FINDING THE DIMENSION FOR A GIVEN FRACTION

In order to answer questions about finding the smallest dimension "d" such that a given fraction of information is preserved, the already explained PCA algorithm is altered to include this functionality. A method `PCA_threshold(X, alpha)` is defined. The main difference lies in the fact that it returns a tuple containing three values (Z_d, p_d, d) with d being the dimension found. The result p_d is the actual fraction of total variance for the new dimension d. The following Listing 7 contains the part that is different to the above PCA implementation. It incrementaly increases the dimension d until the resulting fraction of total variance is just large enough i.e.: $p_i >= alpha$. This dimension d is stored, used for further calculations and later returned.

Listing 7: *projection to lower dimension with selected principal components*

```
... # previous PCA steps

  # calculate number of reduced dimensions based on targeted fraction
    of total variance
  d = -1
  p_d = -1

  for i in range(n):
    p_i = np.sum(variance_contributions[:i]) / np.sum(
    variance_contributions) # fraction of total variance
    if(p_i >= alpha):
      d = i
      p_d = p_i
      break
  if d==-1 or p_d==-1:
    print("fraction total variance calculation failed")
    return None

... # remainding PCA steps
return (Z_d, p_d, d)
```

2.7. Visualizing the reduced data

For visualizing the reduced data for interpretation the libaray pyplot from matplotlib is used. However the reduced data has 40 dimension, which is hard to visualize in a human readable format directly. To achieve a human readable view of the data we used the t-SNE dimension reduction method from SKlearn.manifold library to further reduce data to 2 dimension allowing pyplot to construct 2 dimension plot.

Listing 8: *t-SNE plot: Get data*

```
import sklearn.manifold
reduced = PCA(mat['X'],40)[0]
X_embedded = sklearn.manifold.TSNE(n_components=2).fit_transform(
    reduced)
```

Beginning with obtaining data from aforementioned PCA method, then perform transformation in t-SNE.

Listing 9: *t-SNE plot: Setup*

```
fig = plt.figure(figsize=(10, 8))
plot = fig.add_subplot()
color_arr = ['#e6194b', '#3cb44b', '#ffe119', '#4363d8', '#f58231', '
    #911eb4', '#46f0f0', '#f032e6', '#bcf60c', '#fabebe', '#008080', '#
    e6beff', '#9a6324', '#fffac8', '#800000', '#aaffc3', '#808000', '#
    ffd8b1', '#000075', '#808080']
```

The next step is setting up the plot dimension and add a subplot that is going to show the data to the main figure. Implementing 20 different colors representing 20 different objects by implementing array of colors in form of Hex.

Listing 10: *t-SNE plot: Insert data*

```
for i in range(20):
  plot.scatter(X_embedded[(i*71):(i*71)+71, 0], X_embedded[(i*71):(i
    *71)+71, 1], c=color_arr[i], marker='o', label='Data Points')
```

Then sketch the plot 20 times with different color. The data will be fetch 72 data points each loop for each object representation.

Listing 11: *t-SNE plot: Make plot*

```
plot.set_xlabel('X')
plot.set_ylabel('Y')
plt.show()
```

And finally showing the plot with axis name. This is one of the simple way to visualize the reduced data in 2 dimension form with different colors for better understating.

3. Result

In the following the results of the individual experiments is shown.

3.1. Eigenvalue profile

Figure 1: *A 2D plot showing all 1024 Eigenvalues in the order produced by the np.linalg.eig function*

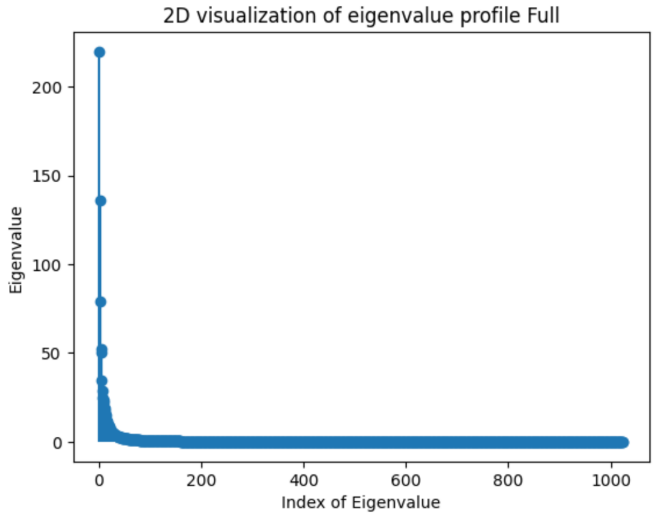
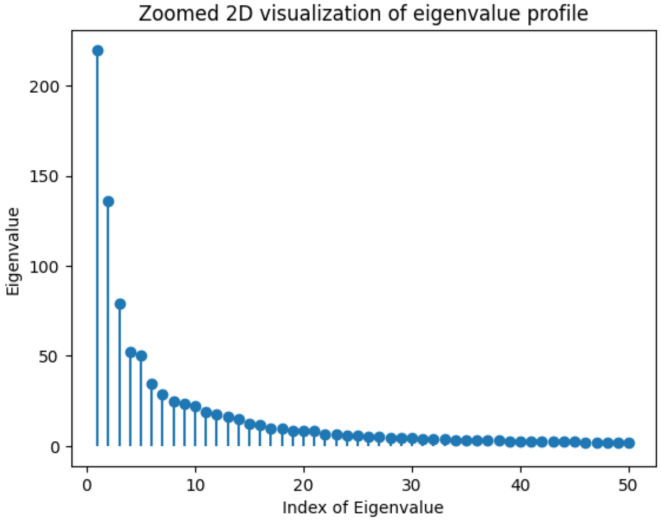


Figure 2: *A zoomed in 2D plot of the first 50 eigenvalues*

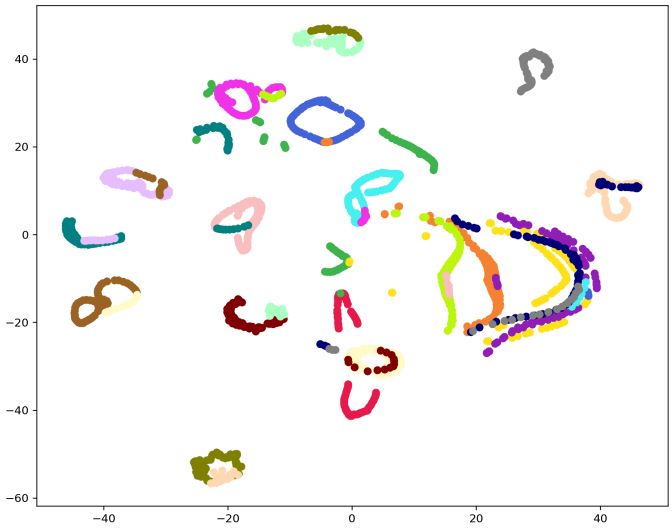


3.2. dimensionality for given fraction of total variance

Target	Solution	No.dimension
0.9	0.9020	40
0.95	0.9505	84
0.98	0.9801	175

3.3. Reduced data

Figure 3: *A 2D plot of the dimensionality reduced data of 20 distinct objects*



4. Discussion

The reduced data will have a different number of columns from original. In this experiment, number of total pixels of each photo has been reduced to 40 from the original 1024, while number of photos is maintained. Thus, can be interpreted as lossy image downgrading but in a way to maximize the variability for the 40 features. Reducing the number of photo and maintain number of pixel is possible by undo transpose when obtaining the U_d , and the eigenvalue, vector, and fraction of variance will differ from one to another.

Another observation is the fact that the individual principal components contributions to the fraction of total variation drops quickly. This sudden drop in "importance" for each feature is well visualized by Figure 1 as well as the zoomed in version Figure 2. It is clear to see visually that for higher fractions of total variance a very high amount of dimensions has to be selected. But on the other hand it also shows that a 90% of the total variance is included in only $40/1024 = 3.9\%$ of dimensions. This is also shown by the fact that to achieve an increase in information retention from 0.95 to 0.98 the number of dimensions needed almost doubled from 84 dimensions to 175 dimensions.

One of our t-SNE observations is that similar image will be plotted in the graph close together. It is obvious that most of the clusters in the graph are arranged in curve line, this is an consequence of the fact that one image is just 5 degrees difference from another one. Therefore, most of the cluster are arrange in concave line, even some are arranged in a circular line.

On the other hand, there are some data points that is are not in clusters, these are most likely images of objects that appear very differently from certain angles. One could imagine such an object as e.g a thin plane. From certain angles there might be almost no visible object, and from other angles it will look very similar.

Originally, it is not possible to show the image with reduced data set. However, with implementation of specific function, the image creation is possible without reconstruction of data. The function finds and display the original image for a given t-SNE data point by using the same index on the original data set. We did so by first identifying an interesting cluster (object) manually. We

chose the object with the index $i = 3$ in the data set as it forms a distinct cluster but also has a few outlier images. The embedding for the object 3 can be seen in Figure 4.

The figure Figure 5 clearly shows that our initial idea on outliers proved to be true and the object indeed looked very distinct from an different viewing angle, therefore explaining some t-SNE embedding data points to be separated from cluster.

Figure 4: *t-SNE plot for one object*

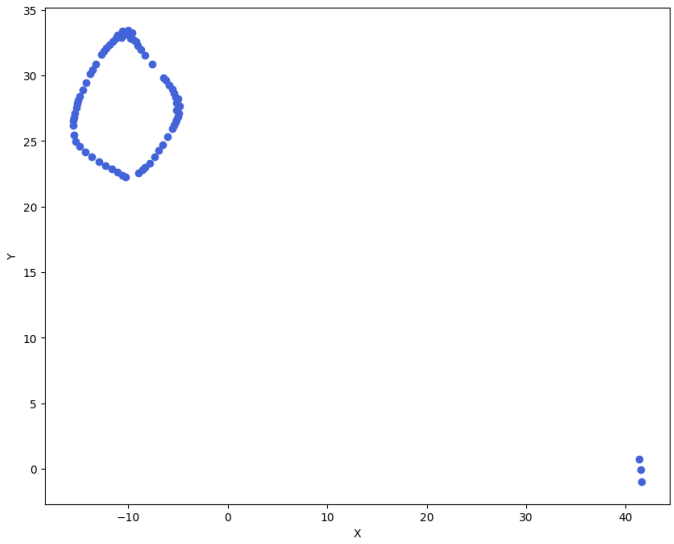
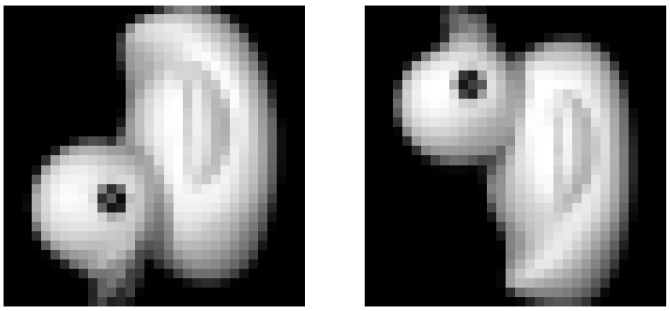


Figure 5: *visualization of object 3. left: cluster image, right: outlier image*



Listing 12: *Visualize image without reconstruction*

```
objectIndex = 3
cluster_centers = [[-10,33], [40,0]]
plt.figure(figsize=(10, 5))
# display original images at cluster_centers of object side by side
for i, center in enumerate(cluster_centers):
    plt.subplot(1, len(cluster_centers), i + 1)
    plt.imshow(mat['X'][np.argmin(np.linalg.norm(X_embedded - center,
   axis=1)[objectIndex*71 : objectIndex*71+71])].reshape(32,32), cmap=
   'gray')
    plt.axis('off')
plt.show()
```

5. Contributions

In total both of us worked the same amount of time and effort [50/50]. Tobias did the PCA and PCA_threshold methods as well as the observations on the object 3. The methods and implementations on these are also described by Tobias as well as corresponding discussion section. Thanakit worked on the t-SNE and introduction and t-SNE Experiments implementation and discussion. Both of us worked together on putting this report together in fair amounts.

References

[1] Rukshan Pramoditha. "11 Different Uses of Dimensionality Reduction". In: (2021). URL: https://towardsdatascience.com/11-different-uses-of-dimensionality-reduction-4325d62b4fa6#:~:text=Dimensionality%20reduction%20finds%20a%20lower,reduction%20helps%20to%20mitigate%20overfitting..

[2] Simplilearn. "What is Dimensionality Reduction? Overview, and Popular Techniques". In: *AIMacine Learning* (2023). URL: https://www.simplilearn.com/what-is-dimensionality-reduction-article#:~:text=Dimensionality%20reduction%20brings%20many%20advantages,because%20you%20have%20fewer%20data.