Assignment 6: Reduction of dimensionality and recognition

Machine perception

2022/2023

General instructions: The assignment is composed of the compulsory tasks and optional tasks (denoted by ★). To approach the assignment defense, you are required to complete at least the compulsory parts. Successfully defending the compulsory part will give you a maximum 75 points out of 100. At the defense, the obligatory tasks **must** be implemented correctly and completely. If your implementation is incomplete or has major errors, you will not be able to successfully defend the assignment. You can choose arbitrarily among the optional tasks to reach the 100 points. The number of points for an optional task is given in the task description. It is possible to obtain more than 100 points on individual assignment. However, the maximum overall points obtained from the 6 assignments is 600 points.

Required formating and submission: Create a folder `assignment6` that you will use during this assignment. Unpack the content of the `assignment6.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as Python scripts to `assignment6` folder. In order to complete the assignment you have to present your solutions to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the assignment defense as well. The code must be submitted on the e-classroom **before** the defense.

COMMON ERRORS AND DEBUG IDEAS

- check your x and y coordinates
- check your data type: float, uint8
- check your data range: [0,255], [0,1]
- perform simple checks (synthetic data examples)

Introduction

In this assignment we will first work with a simple 2D example of the direct PCA method. We will then experiment with the dual PCA method in the case of compact face description. More about the theory of PCA and its applications can be found in [1] (section 22.3.1, pages 507 to 515).

Exercise 1: Direct PCA method

The primary purpose of the PCA method is reduction of dimensionality. This means that we want to find such a linear transformation of the input data that will describe the data in a low dimensional orthogonal space (which can also be called a subspace) while losing the minimal amount of information needed to reconstruct the data. A subspace is defined by its basis vectors. Let us first repeat the direct calculation of the basis vectors with the PCA method. Here we assume that the input data is structured as a column vector with size $m \times 1$:

Algorithm 1 : The direct PCA algorithm.

1: Build a matrix $\mathbf{X}$ of size $m \times N$ so that we combine N input entries as a sequence: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$

2: Calculate the mean value of the data: $\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$

3: Center the data: $\mathbf{X}_d = [\mathbf{x}_1 - \mu, \mathbf{x}_2 - \mu, \ldots, \mathbf{x}_N - \mu]$

4: Compute the covariance matrix: $\mathbf{C} = \frac{1}{N-1} \sum_{i=1}^{N} (\mathbf{x_i} - \mu)(\mathbf{x_i} - \mu)^T = \frac{1}{N-1} \mathbf{X}_d \mathbf{X}_d^T$

5: Compute SVD of the covariance matrix: $\mathbf{C} = \mathbf{USV}^T$ The matrix $\mathbf{U}$ is of size $m \times m$, its columns represent eigenvectors of the matrix $\mathbf{C}$. The corresponding eigenvalues are saved in a diagonal matrix $\mathbf{S}$ and are sorted in descending order. The columns in $\mathbf{U}$ therefore represent an orthogonal basis (subspace) while the eigenvalues give us the level of scattering of the data along the corresponding eigenvector.

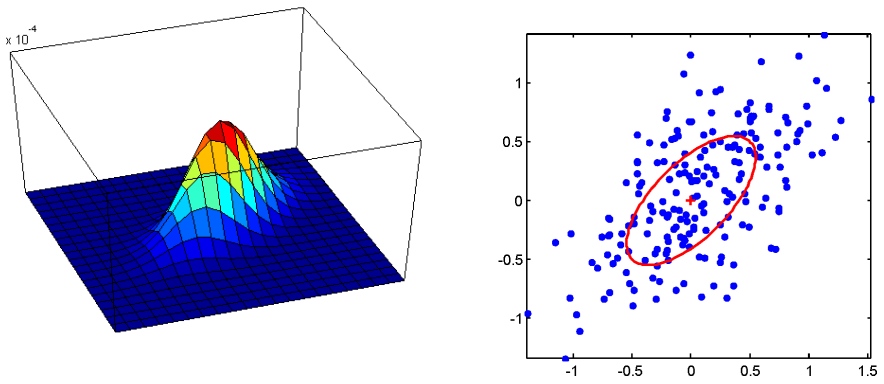
6: A new input entry $\mathbf{x}_q$ is projected in the PCA space (subspace) by first centering it using μ, and then multiply it with the matrix of eigenvectors: $\mathbf{y} = \mathbf{U}^T(\mathbf{x}_q - \mu)$.

7: The transformation from the subspace to the original space is done using equation: $\mathbf{x}_q = \mathbf{U}\mathbf{y} + \mu$.

In the algorithm 1 we can see that in first steps of the PCA method we compute the mean value and the covariance method of the data. These are the two parameters that define a Gaussian distribution. As we will later on refer to this distribution, we should give it a little more attention. We have already introduced Gaussian distribution in Assignment 2, where we have used a special case of such distribution to construct a Gaussian kernel. In that case the distribution had a mean value of zero and a diagonal covariance matrix with equal covariance along the x and y axis. The general formula for a Gaussian distribution is

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^d |\mathbf{C}|^{\frac{1}{2}}} \exp\Big(-\frac{1}{2}(\mathbf{x} - \mu)^T \mathbf{C}^{-1} (\mathbf{x} - \mu) \Big). \tag{1}$$

Figure 1(a) shows an example of a Gaussian distribution, where the values on the z axis display the probability of $p(\mathbf{x}, \mathbf{y})$. Image 1(b) shows an alternative illustration, where we can see a border at which the ellipse contains the 68% of all probability density. Points used to compute the mean value and the covariance used to draw a Gaussian distribution are also shown in the right image.



(a) (b)

Figure 1: a) displays 3D display of a 2D Gaussian distribution, b) shows the same distribution in 2D using an ellipse.

How to calculate eigenvectors and eigenvalues analytically?

Let us briefly repeat how to calculate eigenvectors and eigenvalues of a square matrix $\mathbf{C}$ with size $m \times m$ analytically. Consider a matrix $\mathbf{Z}$, that we get if we multiply a unit matrix of size $m \times m$ with a constant value, e.g., $\mathbf{Z} = \lambda\mathbf{I}$. If we subtract this matrix from $\mathbf{C}$ and we get a determinant 0, we have transformed $\mathbf{C}$ into a singular matrix:

$$|\mathbf{C} - \lambda\mathbf{I}| \equiv 0. \tag{2}$$

The values λ that satisfy the equation above are called *eigenvalues* of the matrix $\mathbf{C}$. For an $m \times m$ matrix, we can compute m eigenvalues λ_i. A product of eigenvalues equals the determinant of the matrix: $|\mathbf{A}| = \prod_{i=1}^{m} \lambda_i$. Each eigenvalue λ_i corresponds to an *eigenvector* $\mathbf{x}_i$. Eigenvectors are mutually perpendicular (the dot product of two eigenvectors is 0).

For eigenvalue λ_i we compute the eigenvector $\mathbf{x}_i$ of size $1 \times m$ by setting up the following equation:

$$(\mathbf{C} - \lambda_i\mathbf{I})\mathbf{x}_i \equiv 0. \tag{3}$$

The solution of the equation gives us the elements of the eigenvector up to the level of scale. That means that the equation can be solved by an infinite number of vectors that only differ in scale. In practice, we pick the value 1 for the first element and then calculate other elements of the vector. In the end, we change the scale of the vector so that its length becomes 1 (unit vector). Unit eigenvector for λ_i is defined as $\mathbf{e}_i = \mathbf{x}_i / ||\mathbf{x}_i||$.

A simple example of a direct PCA

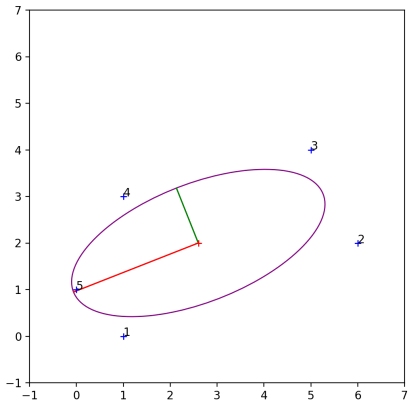
(a) Solve the following assignment for practice: You are given four points $A(3, 4)$, $B(3, 6)$, $C(7, 6)$ and $D(6, 4)$. Calculate the eigenvectors and eigenvalues for the given set of points. You can use SVD. Plot the points and corresponding eigenvectors (originating at the data mean).

(b) Write a script to calculate and visualize PCA from 2D data from the file `points.txt` (the first column contains the x axis and the second column the y axis). Plot the

points and draw the representation of the Gaussian distribution using `drawEllipse` from the supplementary material. Follow the Algorithm 1 to compute the eigenvectors and eigenvalues of the PCA subspace.

(c) The matrix $\mathbf{U}$ contains the eigenvectors that represent the basis of our PCA subspace. Draw the eigenvectors on the plot from the previous task. Their origin should lie at the mean of the data μ. Since both vectors have the length 1, a better way for visualizing them is to multiply each vector with the corresponding eigenvalue[1]. Draw the first eigenvector with red and the second with green.

Question: What do you notice about the relationship between the eigenvectors and the data? What happens to the eigenvectors if you change the data or add more points?



(d) The eigenvalues represent the reconstruction error we incur if we discard the corresponding eigenvector. Plot the cumulative graph of the eigenvalues and normalize it, so the largest value will be 1. From the graph, determine how many percent of the information will we retain if we discard the second eigenvector. To put it differently, what percent of the variance is explained just by using the first eigenvector?

(e) Now remove the direction of the lowest variance from the input data. This means we will project the data into the subspace of the first eigenvector. We can do this by transforming the data into the PCA space, then setting to 0 the components corresponding to the eigenvectors we want to remove. The resulting points can then be transformed back to the original space. Project each of the input points to PCA space, then project them back to Cartesian space by multiplying them by the diminished matrix $\mathbf{U}$.

Question: What happens to the reconstructed points? Where is the data projected to?

(f) For the point $q_{\text{point}} = [6, 6]^T$, calculate the closest point from the input data (using Euclidean distance). Which point is the closest? Then, project all the points (including q_{point}) to PCA subspace (calculated without q_{point}) and remove the variation

[1]Eigenvalue tells us the variance of the data along its corresponding eigenvector. If you multiply the eigenvector with the square root of the eigenvalue, you will visualize the standard deviation along the eigenvector.

in the direction of the second vector. Calculate the distances again. Which point is the closest to q_{point} now? Visualize the reconstruction.

Exercise 2: The dual PCA method

When analyzing image data, the dimensionality can become very large (much larger than the number of elements in the input), the direct method of computing eigenvectors we used previously is no longer suitable. E.g. if we have 10000-dimensional data, this would produce a covariance matrix of size 10000×10000. Here we are close to hitting the limits of computer memory. As the number of data samples N is lower than the dimensionality, we can use the *dual PCA* method. The main difference between this and the direct approach are steps 4 and 5 (compare Algorithm 1 to Algorithm 2).

(a) For our requirements, it is necessary only to correctly calculate eigenvectors and eigenvalues up to the scale factor. Therefore, implement the dual method according to the Algorithm 2 and test it using the data from `points.txt`. The first two eigenvectors should be the same as with the Algorithm 1. The Algorithm 2 gives you a larger matrix $\mathbf{U}$, however, all eigenvectors but the first two equal to zero.

(b) Project the data from the previous assignment to the PCA space using matrix $\mathbf{U}$, and then project the data back again in the original space. If you have implemented the method correctly, you will get the original data (up to the numerical error).

Algorithm 2 : The dual PCA algorithm.

1: Build a matrix $\mathbf{X}$ of size $m \times N$ so that we combine N input entries as a sequence: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$

2: Calculate the mean value of the data: $\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$

3: Center the data: $\mathbf{X}_d = [\mathbf{x}_1 - \mu, \mathbf{x}_2 - \mu, \ldots, \mathbf{x}_N - \mu]$

4: Compute the dual covariance matrix: $\mathbf{C} = \frac{1}{N-1} \mathbf{X}_d^T \mathbf{X}_d$.

5: Compute singular value decomposition (SVD) of the dual covariance matrix, $\mathbf{C} = \mathbf{USV}^T$. Then compute the basis of the eigenvector space as: $\mathbf{U} = \mathbf{X}_d \mathbf{U} \sqrt{\frac{1}{\mathbf{S}(N-1)}}$. Note: it is expected that only the first few of the m or N (the lower one) columns of the eigenvectors will differ from 0.

6: A new input entry $\mathbf{x}_q$ is projected in the PCA space (subspace) by first centering it using μ, and then multiply it with the matrix of eigenvectors: $\mathbf{y} = \mathbf{U}^T(\mathbf{x}_q - \mu)$.

7: The transformation from the subspace to the original space is done using equation: $\mathbf{x}_q = \mathbf{U}\mathbf{y} + \mu$.

Exercise 3: Image decomposition examples

Here we will use the dual PCA method on a computer vision problem. In the supplementary material, there are three series of images. Each series contains 64 images of a face under different lighting conditions. Your task will be to analyze each series using the PCA method.

(a) **Data preparation**: Firstly, we have to formulate the problem in a way that is compatible with the PCA method. Since PCA operates on points in space, we can represent a grayscale image of size $m \times n$ as a point in mn-dimensional space if we reshape it into a vector. Write a function that reads all the images from one of the series transforms them into grayscale reshapes them using `np.reshape` and stacks the resulting column vectors into a matrix of size $mn \times 64$.

(b) **Using dual PCA**: Use dual PCA on the vectors of images. Write a function that takes the matrix of image vectors as input and returns the eigenvectors of the PCA subspace and the mean of the input data.

Note: In step 5 of Algorithm 2 be careful when computing the inverse of the **S** as some of the eigenvalues can be very close to 0. Division by zero can cause numerical errors when computing a matrix inverse. You have to take into account that the matrix **S** is a diagonal matrix and must therefore have non-zero diagonal elements. One way of solving this numerical problem is that we add a very small constant value to the diagonal elements, e.g. 10^{-15}.

Transform the first five eigenvectors using the `np.reshape` function back into a matrix and display them as images. What do the resulting images represent (both numerically and in the context of faces)?

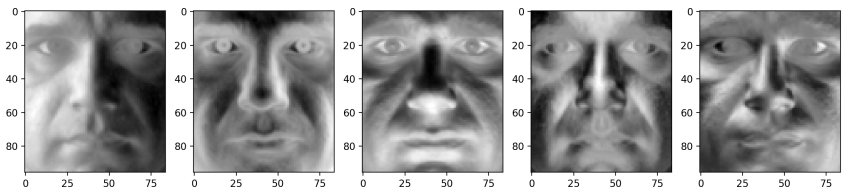


Figure 2: First five eigenvectors calculated from the first series.

Project the first image from the series to the PCA space and then back again. Is the result the same? What do you notice when you change one dimension of the vector in the image space (e.g. component with index 4074) to 0 and display the image? Repeat a similar procedure for a vector in the PCA space (project image in the PCA space, change one of the first five components to zero and project the image back in the image space and display it as an image). What is the difference? How many pixels are changed by the first operation and how many by the second?

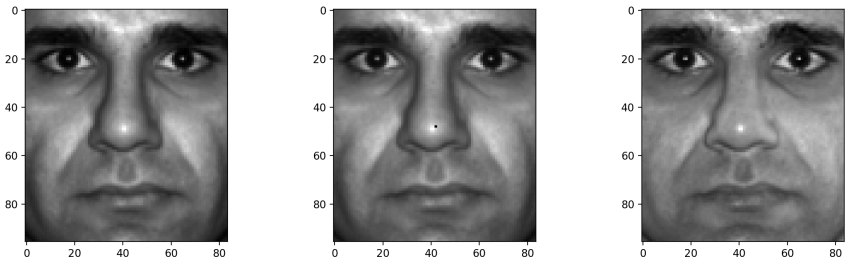


Figure 3: The original image, image with one component changed in the original space, and image with one component changed in the PCA space.

(c) **Effect of the number of components on the reconstruction:** Take a random image and project it into the PCA space. Then change the vector in the PCA space

by retaining only the first 32 components and setting the remaining components to 0. Project the resulting vector back to the image space and display it as an image. Repeat the procedure for the first 16, 8, 4, 2, and one eigenvector. Display the resulting vectors together on one figure. What do you notice?

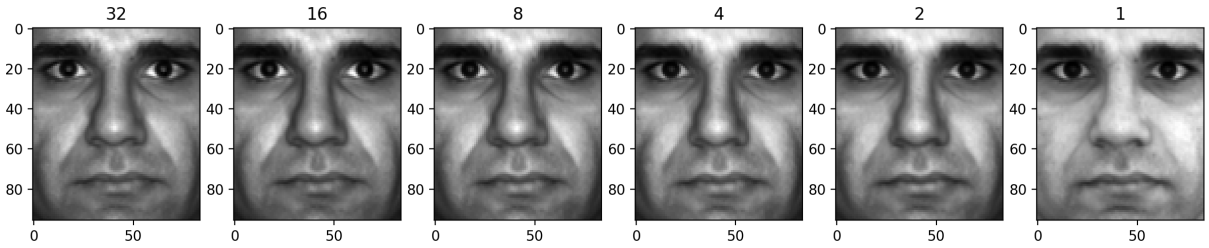


Figure 4: Reconstruction using a different number of components in PCA space, image was taken from the first series.

(d) ★ (10 points) **Informativeness of each component**: Each eigenvector holds information that defines some aspect of the PCA space. By changing the values of a vector in a periodic way, we can observe how different weights for an eigenvector affect the reconstructed image.

Use the second series of images for this task. Take the average photo that you compute based on all images in the series[2]. Project the average image to PCA space. Then, select one of the more important eigenvectors and manually set its corresponding value in the projected vector to some number of your choice. Project the modified vector back to image space and observe the change.

In order to see the changes easier, write a script that goes over a range of values for your selected eigenvector. To smoothly change the values, use `np.sin` and `np.linspace` as well as some scaling factor x to show the differences more strongly. Also, use `plt.draw` in combination with `plt.pause` to display the results as an animated sequence of images.

Hint: We recommend a scaling factor x of around 10.

Modify the script to change two parameters at the same time, in a circular way (i.e. using both `np.sin` and `np.cos`). Experiment with different eigenvector pairs and report your observations.

(e) ★ (5 points) **Reconstruction of a foreign image**: The PCA space is build upon an array of data. In this task, we will check the effect that the transformation to PCA space has on an image that is not similar to the images that were used to build the PCA space. Write a script that computes the PCA space for the first face dataset. Then load the image `elephant.jpg`, reshape it into a vector and transform it into the precalculated PCA space, then transform it back to image space. Reshape the resulting vector back to an image and display both the original and the reconstructed image. Comment on the results.

[2]The value of each pixel equals to the average value of the corresponding pixels in all images. In case of image vectors, we are dealing simply with their average value.

(f) ★ (15 points) **Recognition with a subspace**: PCA subspaces can be used in a simple object recognition scenario. Generally, if an image is similar to the images used to build the PCA subspace, the reconstruction error incurred when projecting to said PCA space should be small. In this task, you will try to implement a simple facial recognition system based on this property.

Use a camera to capture images (10 at the very least) of your face with varying illumination and facial expressions. Resize them to a common resolution and try to align them by eye location. Construct a PCA space from these images. Then, write a script that captures images from your webcam, detects faces in them (you can use something from OpenCV like Haar cascades) and extracts the regions that contain faces. Reshape these regions to correct size, transform them to the precalculated PCA space and back to image space. Then, based on some similarity measure (can be simple L2 norm) and some threshold, decide whether a region contains your face or not. Display the information on the image stream.

Note: You should prepare a video that demonstrates the performance of your system, or run it live during the defense.

References

[1] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, 2002.