Facial Recognition System

Based on Haar Cascade and Principal Component Analysis (PCA) Algorithms

Group Members:
Yachen Gao
Merton Chen
Jinghui Li

BU MET collage Computer Science Department, CS566

Professor Name:
Alexander Belyaev

Spring, 2022

CONTENTS

Abstract

Face recognition has become increasingly important over the last decades. In this project, we will take a deeper look at the two major phases in a regular facial recognition system: facial detection and facial recognition. Haar Cascade is an effective object detection method which can be used for the purpose of the detection stage in our system, while Principal Component Analysis (PCA) algorithm will handle the recognition stage. We will run several testing scenarios to understand how the Haar Cascade algorithm would perform under different conditions or how certain factor could impact its performance. Then we will utilize a sample photo database to test out how accurate the PCA algorithm can be when the conditions of the input training photos are relatively ideal.

Introduction

Nowadays facial recognition has become common use in people's daily life, used on the phones in order to make it easy to use, identify missing people. In real life this analysis is widely used in regional economic development evaluation, pattern recognition, image compression and many other fields. The effective ability of PCA to analyze data and dimensionality reduction of high dimensional and reduce the unrelated principal variables under condition contributes us to successively handle the recognition stages. It becomes one of the most common ways to obtain excellent performance on prediction and redundancy removal. Haar Cascade provides effective object detection that we will be able to accurately detect objects in the first stage of the system. It is a machine learning based approach to distinguish an object from provided resources. The capability of calculating speed become one of the distinguish feature to compute the integral images at a very fast rate. We will dive into the application and principals behind those main algorithms and have a better understanding of facial recognition.

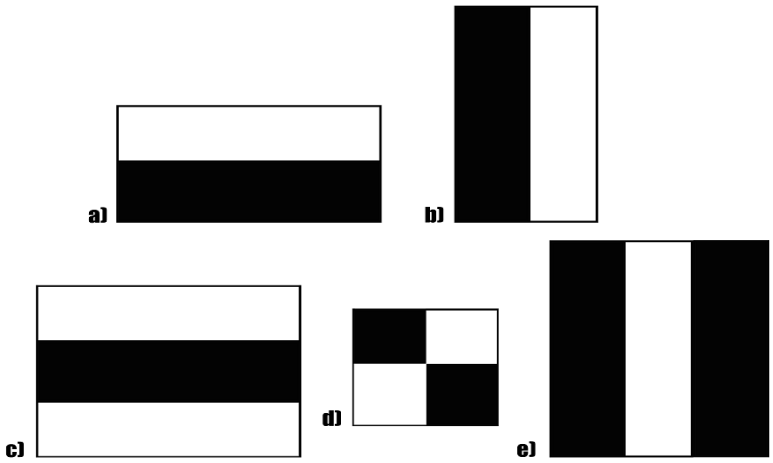
Facial Detection Algorithm: Haar Cascade

One popular method of object detection used today is Haar Cascade, also commonly known as Viola-Jones Face Detection. It utilizes deep learning techniques to distinguish an object, or face, from an image or real time video. In our project we will research how Haar Cascade works with detection of faces from still images.

Haar Cascade utilizes detection of edges and lines. To train the model to be able to detect those features, images are given to train for positive and negative traits (positive means images with faces and negative means images without faces). This paves way for the Haar features to pick out areas where there is an edge or a line. An area with an edge or a line is characterized by a sudden change in intensity of pixels. For example, you would classify the dark parts of the image with values of 1 and the light parts of the image with values of 0. After classifying the different parts of the image, you would find the sum of all the image pixels for light and dark. Next, find the difference between light and dark. A value closer to 1 means that an edge is detected and present.

0.4	0.5	0.6	0.3	0.2	0.1
0.6	0.6	0.6	0.3	0.2	0.1
0.6	0.6	0.7	0.4	0.3	0.2
0.5	0.6	0.8	0.5	0.4	0.3
0.5	0.5	0.7	0.4	0.3	0.2

The Haar features will need to be classified for the whole image. This means that the Haar feature must travel pixel by pixel, top left to bottom right, to classify everything in the image. Different sets of rectangles are used in combination to calculate features on the image like the ones illustrated below.



For example, two rectangle features find edges in a horizontal or vertical direction and three rectangle features find if there are dark regions surrounded by light regions or vice versa. To reduce the number of calculations and work needed to find the Haar features, a concept known as integral image can make the same operation with less work.

An integral image is a concept that uses the original image to create pixels that are the sum of all the pixels left and above the original image. Using an integral image will reduce the total time complexity gradually. All the information about the model and concepts so far details the features and representations of the image. The next step is the actual implementation for image analysis.

For all the features that have been gathered, a feature selection technique is needed to properly select good features and remove inaccurate features. Adaboost is a boosting technique that allows for the selection of subsets from the huge set gathered from an image. Adaboost creates weak learners that reduce the total number of features and produce a low error rate as well. For example, a set of 180,000 features can be reduced to a set of 6,000 features using Adaboost.

After the set of features have been narrowed down by Adaboost, Haar Cascades uses a technique called Attentional Cascade to run training on images in a more efficient and effective way. The technique involves running features in stages. Each stage tests for features starting from simple to more complicated. If a feature is not detected in a stage, it is removed from the process while a feature that is detected will move on to more complicated stages. This reduces a lot of images that need to be processed by removing irrelevant windows.

Haar cascade can be accessed through OpenCV and allows one to use their own classifiers for any objects, in this case faces, to train. Haar cascade is an old object detection method that is still widely used today. Haar Cascade's ability to provide accurate results consistently makes it a competitive option against other face detection methods.

Facial Recognition Algorithm: PCA

- ### Mathematics of PCA

The central idea of the principal component analysis is to reduce the dimensionality of a data set consisting of many interrelated variables. PCA can be thought of as unsupervised learning problem, the process of gaining principal components from a raw dataset can be simplified in five parts:

1: standardization of data (data preprocessing: mean normalization)
2: computation of covariance matrix
3: calculation of the eigenvector and eigenvalue
4: selection of the number of principal components.
5: multiplication of principal components with original data to create the newly transformed data set.
(A 2D facial image can be represented as a 1D vector by concatenating each row into a long thin vector. for example, we have M vectors of size = rows of image * columns of image representing a set of sampled images)

1: We will assume there are no train sets. Different sets have stored different data.

Each mean of the data will be represented as $u_j = \sum_i x_i^{(j)}$ and then use each set mini this mean and then divide a. The reason for doing this is that each matrix has a different value of data, we need to make sure the difference will be united.

2: Calculate the covariance of the matrix:

This property is especially valuable because covariance matrices are symmetric matrices whose eigenvectors are orthogonal. To generate the shadow space, we need a set of basis vectors. If there is no orthogonal relation between the basis vectors, when a matrix is projected into the space, it is likely to cause information aliasing, resulting in data loss. For the formula of the symmetric matrices is $\Sigma = \frac{1}{m} T T^T$ This is the symmetric matrix, and it is also called square matrix.

3: Calculation of the eigenvector and eigenvalue:

These parts involve a broad knowledge of linear algebra, and we need to calculate the eigenvalue of the symmetric matrix. For every matrix $A = UV\Sigma^T$ Matrix U has AA^t eigenvalue and matrix B has A^tA eigenvalue. When those two eigenvalue are in the same matrix, we will have the formula $A = UV\Sigma^T$ which they can be represented by the same eigenvalue. The bigger this eigenvalue is, the bigger the database size is.

4: Selection of the number of principal components

eigenvalue has a positive correlation with the eigenvector, we will have better ability to create basis and when we use the eigenvalue and the vector to form a facial space, we will have the path to the recognition

5: multiplication of principal components with original data to create the newly transformed data set. when we the dimension of n of the real-life data compared to the dimensionality of m realistic data. we need to put those real data to the perspective data. The formula represents the length of the vector so when m is dimension space, the data provided by this promular is the ideal data we want to receive. When A is gained from a vector and symmetric matrix, Therefore, the projection can be realized only by multiplying the matrix space of real data after transposing the projection space. $U = \frac{AA^T B}{A^T A}$

Those are some definitions will help us to understand the terms.

(covariance): Covariance issued to depict the relationship between two variables. If the covariance is positive then two variables tend to move in the same directions. In other cases two variables will move to different directions if the covariance is negative. If the covariance is 0 then there is no relationship between those two variables.
(variance): it tells us how far the values are from the mean. It is the spread of the data.
(Eigenvector) is a nonzero vector that only changes the magnitude and direction when subjected to a linear transformation.
(Eigenvalue) if we multiply a non zero vector with the covariance matrix. Then the result will be a scaled version of the initial vector and the scaled value is called eigenvalue.

- **Face Recognition**

Once we have the eigenfaces computed, there are several types of decisions that will be made based on the application we have. It has a specific task in this broad term which is first one is identification where the labels will be obtained. Second one is recognition of a person where it will be decided if a person has already been seen. Third one is the categorization where the face will be assigned to a certain class.

In a space a basis is represented by training vectors. Those vectors are what we were just talking about, which is the eigenvectors computed by PCA. There are several types of decisions we will make based on the application we have.

- We need to identify the labels of individuals
- We need to recognize a person if the individual has already been seen.
- We need to categorize the face to a certain class.

Based on the eigenvector that is computed by PCA. It generates the eigenfaces. Each eigenface can be viewed as a feature, we will recognize those features in the face so that we will be able to tell the facial images by its eigenface weights or coefficients. Each face is transformed into a face space and its component and element will be stored in the memory.

Testing Scenarios for Haar Cascade Algorithm

Cascade Classifier under OpenCV library in python provides a convenient way of utilizing the haar cascade algorithm, so all our testing scenarios in this section will be applied to use the Cascade Classifier. Before we pull the testing photos into the detector, there will be a photo normalization step to reformat, resize and remove the colors of the photos. This step will dramatically reduce the noise and guarantee the best condition of the input photos for the following detection step.

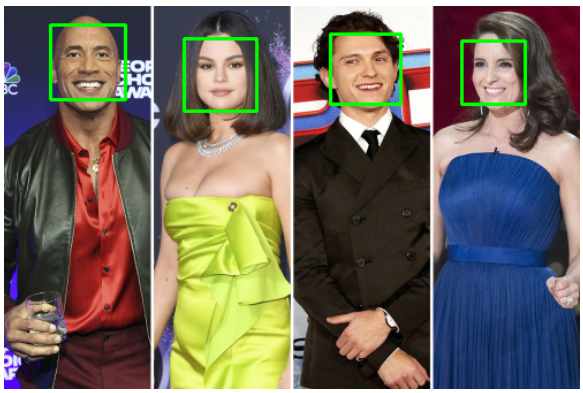
There are three important factors in Cascade Classifier: scaleFactor, minNeighbors and minSize.

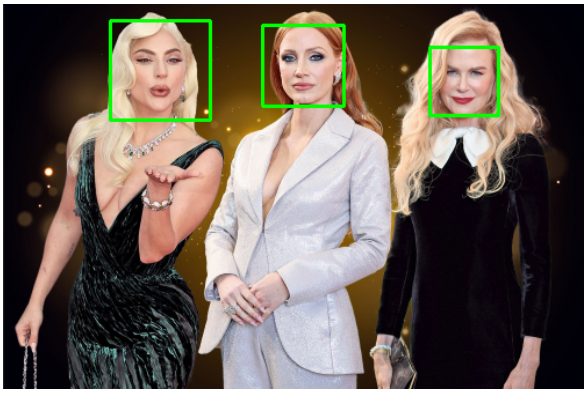
- The scale factor specifies how the image size is modified at each image scale, thus the model has a fixed size for training. If the factor is small, the detection could be slow as it is more precise.
- minNeighbors is specifying how many neighbors each candidate rectangle should have to retain it. In other words, this parameter will affect the quality of the detected faces. Higher value results in less detections but with higher quality.
- minSize represents the minimum possible object size, which means objects smaller than that will be ignored.

Since minSize is straightforward to understand, we will just use (10, 10) to be the constant input across all testing scenarios, the value we are using here is a relative conservative and reasonable number for what we are trying to test.

Test 1 – test for the two input factors: scaleFactor & minNeighbors

We used 11 photos (from Oscar ceremony this year) which contain 51 full faces for testing the algorithm, there are two very important factors in this function, scaleFactor and minNeighbors. If we fill them with reasonable values, below are the examples of how well the algorithm can detect the faces.





Now we would like to test if we change these two factors within a certain range, how much it would impact the detection results:

- Scale factor: 1.01 – 1.20, step by 0.01
- Min neighbors: 5 – 25, step by 1

How many faces were detected		min neighbors																				
		5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
scale factor	1.01	79	76	71	68	66	63	61	57	56	54	54	53	53	52	49	49	47	45	44	44	44
	1.02	65	58	56	53	50	48	46	45	44	43	41	41	41	41	38	38	37	37	37	37	37
	1.03	59	52	49	48	42	42	42	40	39	37	37	36	36	36	36	35	34	34	32	32	32
	1.04	48	45	44	41	41	40	39	37	37	37	36	35	34	32	31	31	30	30	30	30	30
	1.05	47	42	39	36	36	34	34	34	33	33	33	32	32	30	30	30	28	28	28	28	28
	1.06	43	39	39	37	37	37	34	32	32	32	32	30	30	30	30	30	30	29	27	27	25
	1.07	40	35	34	34	33	33	31	31	31	29	29	29	29	28	27	26	26	25	24	24	22
	1.08	41	37	36	35	34	34	33	31	31	30	30	29	27	26	24	24	22	22	19	18	18
	1.09	36	35	35	35	31	30	30	28	28	26	26	23	23	23	23	23	22	20	20	20	18
	1.1	34	34	34	33	32	30	29	29	29	29	27	26	25	25	24	22	21	19	19	18	18
	1.11	34	33	31	31	31	30	29	26	26	25	23	23	21	20	19	19	19	17	17	17	16
	1.12	34	32	32	31	29	29	28	27	25	25	23	22	19	18	18	17	16	16	16	15	15
	1.13	36	32	31	29	29	28	27	25	24	24	23	22	19	18	18	17	17	17	17	17	17
	1.14	33	31	29	28	26	25	24	22	20	20	19	16	16	16	16	16	16	16	15	15	15
	1.15	33	29	29	29	27	26	24	24	22	21	20	20	18	17	17	17	17	17	17	16	15
	1.16	30	29	27	25	24	22	21	21	19	19	17	17	17	16	16	15	15	15	15	15	15
	1.17	30	30	28	25	24	23	23	19	18	18	18	16	16	16	16	16	15	15	15	15	14
	1.18	29	28	27	25	21	21	21	19	18	17	17	17	16	15	15	15	14	14	14	14	14
	1.19	30	27	26	26	25	23	23	21	18	17	17	17	17	17	17	17	16	16	15	15	15
	1.2	30	29	26	25	25	21	21	17	17	17	16	16	16	16	16	15	14	14	14	13	13

Observations from this testing:

- As we increase the scale factor, it will detect less faces
- As we increase the min neighbors, it will detect less faces
- Even a slight scale factor change, can results In big impact

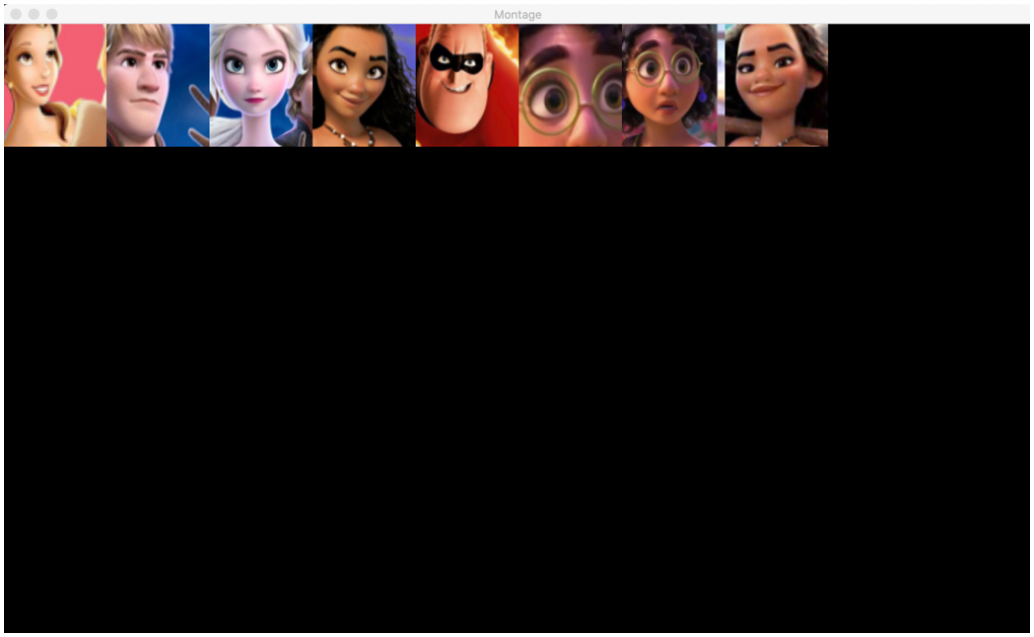
- If the scale factor is too low, there will be a lot of False positives, so we really need to ne cautious with this input.

Below is one example to show the results with False positives: (*Scale factor: 1.01 Min neighbors: 16, total faces detected: 53*)

If we increase the scale factor to 1.05, all the false positives will be gone, but at the same time, some of the faces with darker light, blurred conditions or tricky angles will be gone as well.

Test 2 – test with cartoon faces

We used 10 photos from Disney movies which contain 38 full cartoon faces (both human & animal) for testing the algorithm. (*Scale factor: 1.05, Min neighbors: 16*)
Only 7 were detected, 1 false positive, all human, no animals with a cartoon “human-look face”

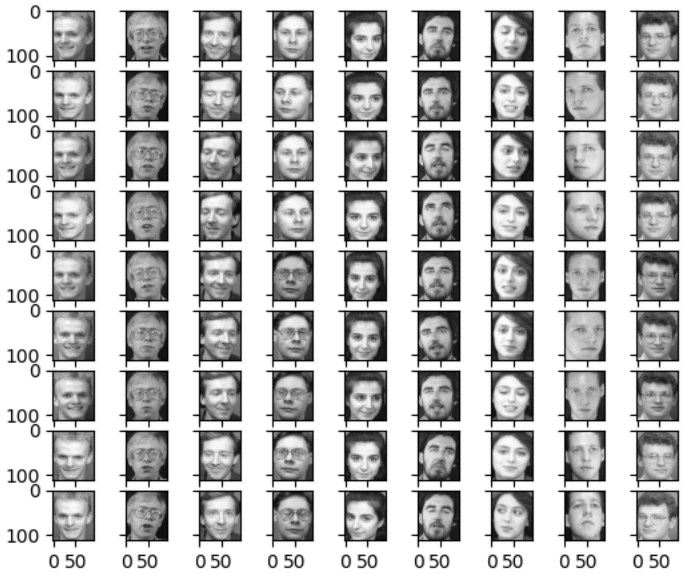


Below are the examples of what type of faces were detected in this case:

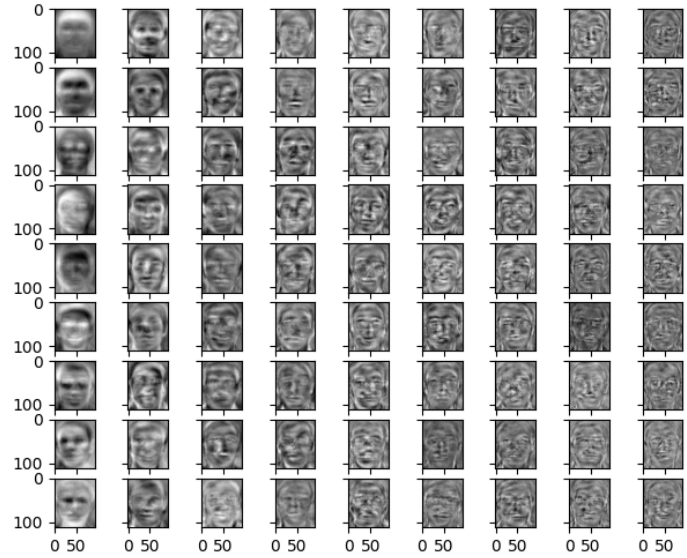
Test 3 – test with animal photos
We used 10 animal photos which contain 34 full faces (cats, dogs and otters) for testing the algorithm, it detected 0 face even we adjusted the factors as Scale factor=1.05, Min neighbors=5, which should be low standards for facial detection, but the animal faces didn't cheat the algorithm successfully.

Testing for PCA Algorithm

This algorithm was applied by using the python library "sklearn.decomposition". We used 400 standard training photos of 40 person, 10 photos per person to test on this algorithm, and all the photos are in the same size and background color with limited extra space other than the face itself.
Our testing plan is to keep one photo from each person as the testing samples, so we have 40 in the testing list now, and the rest of the 9*40=360 photos will be our facial recognition database, we would like to check if the algorithm is able to identify this person's face in the database and grab the correct person's name accordingly.
Due to we have ten photos of each person, so we can have 10 testing groups. We will repeat the above steps for each group and demonstrate the recognition accuracy results in the end.
To get started with the testing, below is one example of how the faces look like in our "database" (use 9 person as an example, while we have 40)



After the algorithm performs the principal component analysis on our dataset matrix, we will get the principal component vectors (eigenface), they are Numpy arrays in the code, but if we transfer it into 2-D, below is an example of how they look:

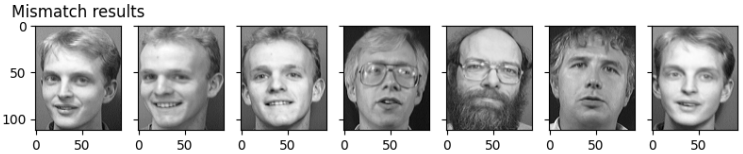


In the end, below is an example of what we got for testing one group, the first row represents the testing photos, the second row is the closest match in the photo database that the algorithm recognized, each person's name is at the top of each photo, so we can compare them side by side. We can see that the algorithm did a good job for recognizing the testing faces with only a few mismatch cases.

Running the 10 testing groups together, we will get this output summary table regarding the accuracy. We can see that the overall accuracy percentage is quite high and consistent.

Testing group	Face regonized correctly	Accuracy
1	39	97.50%
2	40	100.00%
3	40	100.00%
4	39	97.50%
5	39	97.50%
6	40	100.00%
7	39	97.50%
8	39	97.50%
9	39	97.50%
10	37	92.50%
Overall	**391**	**97.75%**

Below are the examples of the mismatch faces, we can see that even they are not from the same person, they do have some similarities visually:



Discussion and Conclusion

Both algorithms we have deep dived and applied in this project have its limitations even they can work quite good with faces/photos in ideal conditions:
For Haar cascade,

- the performance will be impacted dramatically by the factors like “scan” scale and min neighbors, training for the best input factors is important, scale factor is very sensitive to the changes, so we need to be very cautious with picking the reasonable number.
- The input photo’s condition will also play a big role to the success of the results, this algorithm has a relatively weak performance to blurred and dark photos.
- We identified very limited non-front faces even use a low scale factor, and we almost identified zero non-front faces with a regular scale factor. So, this algorithm has a very bad performance with faces with tricky angles to the front.
- The speed of the algorithm can be extremely slow if you use a small scale-factor, so it will be expensive to deal with a lot of photos together. At the same time, the false positive percentage can be over 20% with a low scale-factor.

For PCA,

- PCA is not as readable and interpretable as original features.
- We must standardize your data before implementing PCA. All the categorical features are required to be converted into numerical features before PCA can be applied
- Although PCA tries the best to cover the maximum variance among the features in the dataset. If we don’t select the number of principal components. we will lose the information as compared to the original list of features.
- The condition of the training photos is super important, for example, if the training photos’ background is in different colors (even black and white), this algorithm won’t be able to match the faces with the correct person based on the training results with bias.
- All training photos need to be in the same size, which adds some challenges to the face cutting from the real-world photos while collecting the training samples, because no ideal width/length ratio will work the best for all faces/photos.

References

- Behera, G. S. (2020, December 29). *Face detection with Haar Cascade*. Medium. Retrieved April 17, 2022, from https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08
- *Face detection using Haar Cascades¶*. OpenCV. (n.d.). Retrieved April 17, 2022, from https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html
- https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08
- https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

- https://machinelearningmastery.com/face-recognition-using-principal-component-analysis/
- M.A. Turk and A.P. Pentland, “Face Recognition Using Eigenfaces”, IEEE Conf. on Computer Vision and Pattern Recognition
- http://staff.ustc.edu.cn/~zwp/teach/MVA/pcaface.pdf
- https://melaniesoek0120.medium.com/principal-component-analysis-pca-facial-recognition-5e1021f55151
- Behera, G. S. (2020, December 29). Face detection with Haar Cascade. Medium. Retrieved April 17, 2022, from https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08
- Face detection using Haar Cascades¶. OpenCV. (n.d.). Retrieved April 17, 2022, from https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html
- [2] K. I. Diamantaras and S. Y. Kung, “Principal Component Neural Networks: Theory and Applica- tions”, John Wiley & Sons,Inc., 1996.
- [3] Alex Pentland, Baback Moghaddam, and Thad Starner, “View-Based and Modular Eigenspaces for Face Recognition”, IEEE Conf. on Computer Vision and Pattern Recognition, MIT Media Laboratory Tech. Report No. 245 1994