BLG 453E Homework - 5

Due 10.01.2019 23:59

Policy: Please do your homework on your own (Do not copy paste your solutions from the internet or your friends). The code and the report you submitted must be your own work. All code must be implemented using **Python 3.5+** programming language and **OpenCV Python wrapper**. If you use other version of Python and your code give an error because of Python version you will get 0 point.

Include necessary files with your homework. Do not use absolute file paths. The deadline for this assignment will not be postponed.

You do not have to use PyQt5 in this homework but if you use PyQt5, you will get 50 extra points

For your questions: albay@itu.edu.tr

1. **Lucas-Kanade**

   In this problem, you are required to implement your own "Lucas-Kanade OF" algorithm. Image sequence is given in `traffic_sequences` folder. A sample image is shown below:

Figure 1: Sample image from `traffic_sequences`.

Calculate optical flow at 50 pixel intervals. Do not use bult-in optical flow function but you can use 2d convolution function in Python libraries such as scipy.

2. **Eigenfaces**

In this problem, you are going to implement famous "eigenfaces" method that is used for face recognition. You are going to apply PCA on face images and find the principal components of the high-dimensional vector space of faces.

Principal Component Analysis:

(a) Prepare a set of face images using the images in the "Face_database" folder. Each image will be treated as one vector, simply by concatenating the rows of pixels in the original image, resulting in a single row with $r \times c$ elements. N is the number of the images in the database. We store all images of the set in a single matrix $\boldsymbol{X}_{(N\times(r\times c))}$, where each row of the matrix is an image.

(b) Compute the average image $\boldsymbol{\mu_X}$ and then subtract from each original image in $\boldsymbol{X}$: $\boldsymbol{z}_i = \boldsymbol{x}_i - \boldsymbol{\mu_X}$. Define $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]^T$

(c) Calculate the eigenvectors and eigenvalues of the covariance matrix $\boldsymbol{S} = \boldsymbol{Z}^T\boldsymbol{Z}$ using the builtin function "svd" in Python. Each eigenvector has the same dimensionality as the original images, and thus can itself be seen as an image. The eigenvectors $\boldsymbol{B} = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ of this covariance matrix are called "eigenfaces", where d is the number of eigenvectors. They are the directions in which the images differ from the mean image. Plot the mean image and five eigenvectors with largest associated eigenvalues.

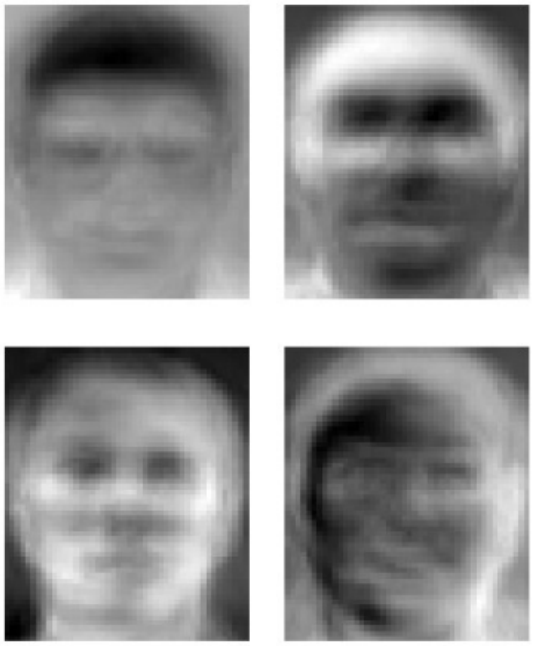

Figure 2: Sample of eigenvectors obtained from face images.

Face Recognition:

(a) To create a new image $\boldsymbol{p}$: Select an image from the image database and add Gaussian noise with a small variance to the selected image.

(b) Transform the created image $\boldsymbol{p}$ into basis $\boldsymbol{B} = [\boldsymbol{b}_1 \cdots \boldsymbol{b}_d] : \hat{p}^j = \boldsymbol{b}_j^T(\boldsymbol{p} - \boldsymbol{\mu})^T$

(c) Transform the images, call them $\boldsymbol{y}_i$, $i = 1, \cdots, N$, that are in the database into basis $\boldsymbol{B} = [\boldsymbol{b}_1 \cdots \boldsymbol{b}_d] : \hat{y}_i^j = \boldsymbol{b}_j^T(\boldsymbol{y}_i - \boldsymbol{\mu})^T$.

(d) Find the vector $\boldsymbol{y}_i$ that is closest to $\hat{\boldsymbol{p}} = [\hat{p}^1 \cdots \hat{p}^d]^T$

(e) Return face image $\boldsymbol{x}_i = \boldsymbol{\mu} + \boldsymbol{B}\hat{\boldsymbol{y}}_i$

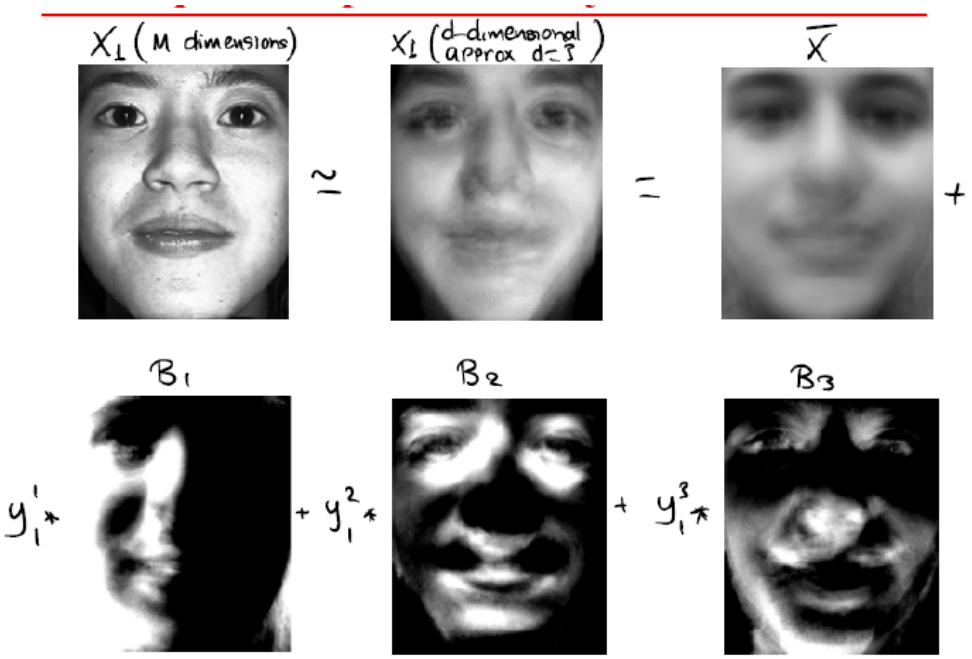

Figure 3: Face recognition example.