TRINITY COLLEGE DUBLIN
School of Computer Science and Statistics

YAHONG ZHU (ID:21333370)

Convolution

i. First, we load an image from the lfw dataset, then using cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb) function to convert the image from RGB color space to YCrCb color space. The Y channel of YCrCb represents the light intensity, therefore we slice the YCrCb image into 3 components Y, Cr, and Cb, then extract Y and plot the image which is shown in the right of Figure 1.

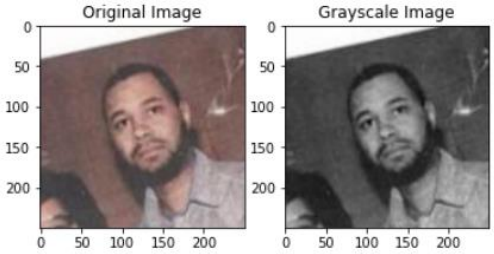

Figure 1 Grayscale Luminance Image

ii. We use cv2.getGaussianKernel function to compute two 1D Gaussian kernel with the same size and sigma. In this case, we choose size=5*5 and sigma=3, then calculate the dot product of these two vectors to obtain the 2D Gaussian kernel. Applying np.gradient function to the 2D Gaussian kernel to compute the derivatives of both horizontal and vertical direction of the kernel. Then, using cv2.filter2D function to apply these filters separately to the grayscale image by convolution. Figure2 shows the results of applying horizontal and vertical derivatives of the Gaussian filter to the grayscale image.

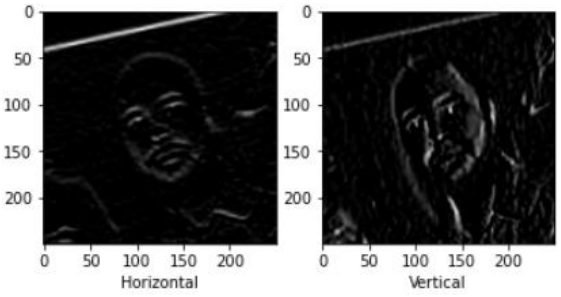

Figure 2 The result of horizontal and vertical derivatives
of the Gaussian filter applied to the image

We add up these two kernels (both horizontal and vertical) to obtain the derivatives of the Gaussian filter and apply it to the grayscale image to get the resulting gradient images(Figure 3).

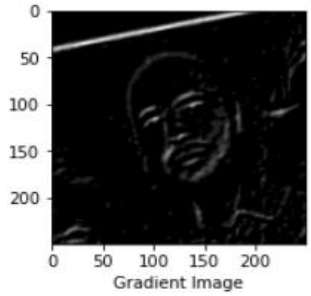

Figure 3 The resulting gradient images

PCA

i. The images in lfw dataset are raw, which indicates that there are some background noises in every image. Some of them even contain more than one human faces, which will cause some error when we use them as inputs of some algorithm. Compared to processed facial images, it has a mass of redundancy which was not useful while using this dataset to train a neural network, and it's time-consuming when we compute the pixel data.

ii. The read_directory function is used to load the dataset and subsample the image by pyrDown function in the loop (size 62×62), converting them to grayscale images by the convertToGrayscale function and then flattening the images into 1D vectors and appending them to the matrix. The shape of this matrix is (3844,1054).

iii. To get the eigenvalues and eigenvectors, we need the mean facial image first by calculating the mean value of the matrix using np.mean function. We reshape it from size 3844×1 to size 62×62 to plot the mean face shown on the first plot of Figure 4. We subtract the mean values from the original image matrix, using the result to compute its covariance matrix by np.cov function. Based on that matrix we calculate its eigenvectors and corresponding eigenvalues. We then sort the eigenvalues from small to large and take the corresponding eigenvector from its end to its beginning. From Figure 4, it can be seen the first eigenface contains the most features of a human face, whereas the middle and the last one don't show any shape of the face. This transform is similar to the Batch Normalization in CNN.

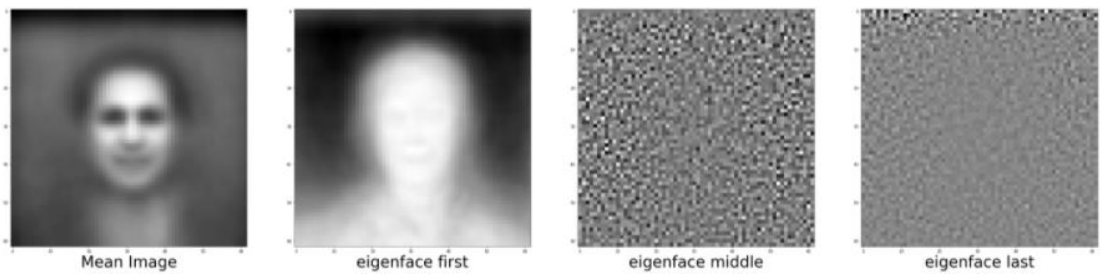

Figure 4 Mean face and Eigenfaces of the first, middle and last eigenvectors

Figure 5 is the plot of the eigenvalues. It illustrates that the eigenvalues decrease with the increase of the principal components of the image. It flattens out when

the amount of principal components is around 500.

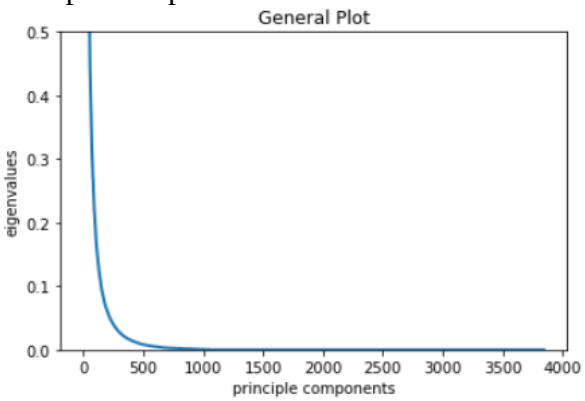

Figure 5 Plot of the eigenvalues

iv. The first image shown on Figure 6 is the original input image. We find the first eigenvalues which value is larger than 0.01 using a loop and extract its index M. Then we use this index as the value of k to obtain the corresponding eigenvector. We project the flattened image into that eigenspace by calculating the dot product of the transpose eigenvector matrix and that image vector. We reconstruct the image using the dot product of the eigenvector and the projection matrix of that image. After reshaping the result into size 62×62, we have the second plot in Figure 6. The rest two plots show a small value of k and a large value of k.

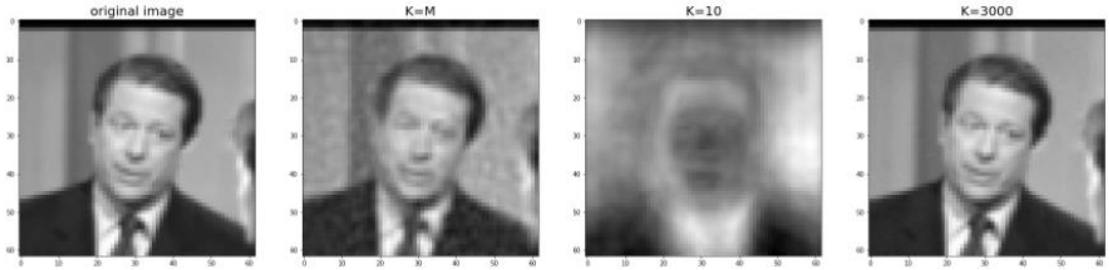

Figure 6 Reconstructed images

As we can see from the result, the reconstructed image holds most of the features of the original image when k=M. When we use a small value of k, the reconstructed image shows a blurry face that doesn't contain many features of the original one. With respect to the large k value, we get a detailed facial image that contains a huge amount of features which is very close to the original image.