SECOND INTERIM REPORT

Dmytro Lutchyn, Nazarii Kuspys

With the increasing need of high-level protection, person identification, and simplification of life, the problem of face recognition became more and more relevant. The progress in science and technologies made it possible to deal with it in various ways. One of which, and the one we are going to be using, is face recognition using principal component analysis (PCA).

Our primary goal is to create an application that will take in a photo of a person and identify them if they are in the dataset or say that the identity of the person on the photo is unknown. The dataset is intended to consist of students and faculty in our university, so whenever you feed a photo to the application it is going to say is the human on it is studying or working in UCU.

We also consider implementing face detection, so we need not feed only the picture of a face, but also a full-length photo of a person. Although it is a great feature, it does not relate on the material we have learned on Linear Algebra course. Face detection can be implemented using convolutional neural networks.

Related work and possible approaches

Based on the literature review that we have made on this topic, we determined that there are quite a few ways of solving the problem of face recognition like linear regression classifier (LRC), principal component analysis (PCA), convolutional neural networks, and many more.

The latter one, building a convolutional neural network model that first needs to be trained to detect distinct features of one’s face, transform them to numerical values, then process input image and compare the results of computations with the results of the person stored in the database (for example, by computing the triple loss function).

PCA, pros and cons

Principal Component Analysis (PCA) method lives up to its name. It is based on selecting the most distinct eigenfaces (principal components) and with their help representing person's face. In simple words, we select some number of most significant features of a face given the dataset and try to represent other faces as their linear combination.

Pros of this method are significant. Using PCA is much more efficient in terms of speed and memory. Typical photo is 100x100 or more pixels, thus it means that the input dimensionality (number of pixels we rely on) is 10,000. Performing computation on data of this volume is much more resource consuming than selecting couple dozens of general face features (principal components) and working with them. Besides dimensionality reduction, PCA also removes correlated features, that leaves us working with only independent set of features, which increases efficiency.

There are a few cons of using PCA, like loss of the information. As we select only certain eigenfaces (principal components) we discard other, thus some features can be lost. Principal Component Analysis can be bad operating with different face expressions, lighting, head position/rotation, image quality, etc. that are not in the initial training database.

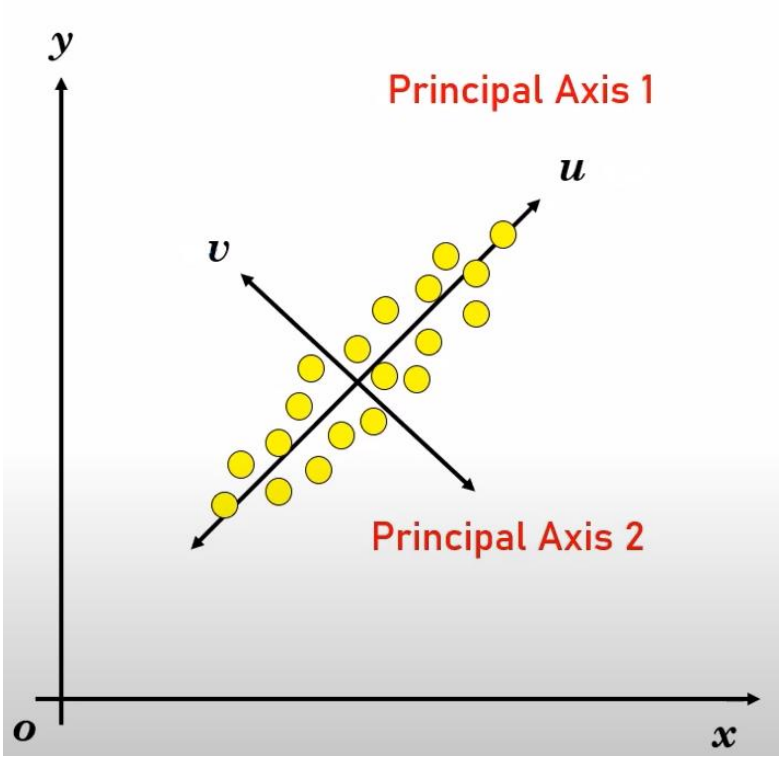
Theory behind PCA

Let us have a set of ***n*** images with the size *MxN*. At the moment each picture is represented by MxN pixels. If we do not reduce dimensionality of each image, the computations will take a lot of time and memory. Our goal is to reduce the number of features each photo is dependent on. For this, we first need to transform every image into a row vector of size *1xMN* (we reshape 2D image into 1D vector)***.*** So, if we have n images, we stack their vector representation on one another and get the input matrix X, which represents all of our photos.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} x_{11}\ x_{12}\ x_{13}\ \ldots\ldots x_{1MN} \\ x_{21}\ x_{22}\ x_{23}\ \ldots\ldots x_{2MN} \\ x_{31}\ x_{32}\ x_{33}\ \ldots\ldots x_{3MN} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{n1}\ x_{n2}\ x_{n3}\ \ldots\ldots x_{nMN} \end{bmatrix}$$

Here each row represents a corresponding pixel intensity value of the image. Next, we need to perform the PCA to find principal components that will best represent images. For this let us use the 2-dimensional example.

Say we have data in x-y axis, and we want to project it into the v-u axis. The covariance between the values in v-u axis is very small, it means that the impact of value in one axis on the value in other axis is very low or zero (the data in de-correlated). This way, if we can reduce the effect of small variations along the v axis, it can be assumed that data points lie on u axis only.



In order to find the principal axes, we need to construct the covariance matrix. In x-y system the covariance is strong, and the diagonal entries (variance) is not dominant. While in case of v-u axis the variance will be strong and covariance between the values (non-diagonal entries) will be zero or of very low significance.

$$Covariance\ Matrix = \begin{bmatrix} var(x) & cov(x,y) \\ cov(x,y) & var(y) \end{bmatrix}$$

To get the new v-u axis we need to diagonalize the covariance matrix for x-y (make variance more significant compared to covariance between the values).

Thus, when talking about set of images, we want to project the pixel values in MN-dimensions onto fewer dimensions, so the covariance between the axes was as low as possible.

Before constructing a covariance matrix, we need to normalize the data, which means subtracting the mean of the images from every photo. This will help get rid of common features and leave us only with distinct ones. The mean is a 1xMN vector, each entry of which is a mean value of corresponding values in X (image matrix).

$$X_m = X - mean$$

After normalization we can construct the covariance matrix Q using the following formula:

$$Q = \left(\frac{X_m^T X_m}{n-1}\right)$$

The resulting covariance matrix will be of the size MNxMN. As we can see, it is a good idea to rescale every image to appropriate size (for instance, 150x150 to reduce memory usage).
This covariance matrix is for old axis system, where the non-diagonal values are strong (the data is correlated). To get the new principal axes, we need to decorrelate the data, thus diagonalize the matrix Q. This could be performed using orthogonal transformation matrix P.

$$P^T Q P = A$$

The matrix A is a diagonal matrix of size MNxMN with eigenvalues of covariance matrix Q as its diagonal entries. This means that all non-diagonal values are zero, which means that the matrix is decorrelated.

The matrix P can be constructed by eigenvectors of Q. These eigenvalues are also called eigenfaces, as they are used to represent the faces. The size of matrix Q is MNxMN, therefore there are MN eigenvectors of a size 1xMN. The P projection matrix will look like this.

$$P = [P_1\ P_2\ P_3\ \ldots\ldots\ldots P_{MN}] = \begin{bmatrix} P_{11} & P_{21} & P_{31} & & P_{MN1} \\ P_{12} & P_{22} & P_{32} & \ldots\ldots\ldots & P_{MN2} \\ \vdots & \vdots & \vdots & & \vdots \\ P_{1MN} & P_{2MN} & P_{3MN} & & P_{MNMN} \end{bmatrix}$$

Now we need to choose and sort the appropriate eigenvectors. Usually, we take eigenvectors that correspond to the eigenvalues that are greater than 1, however you can choose any sufficient number between 1 and MN, for example, 32 or 64.

As mentioned above, we can now represent each face in dataset as a linear combination of chosen eigenvectors. So, the face will be represented as a weighted combination of different eigenfaces.

$$(w1, w2, \ldots, w_k) = (x - m)u1, \ldots, (x - m)u_k$$

$$img = m + w1 * u1 + w2 * u2 + \cdots + w_k * u_k$$

Here, (w1, …, wk) are the weights of corresponding eigenvectors u1, …, uk, m - mean and x is an operating image. We add mean to the image to compensate the normalization we have done.

Now we try to find the matrix that represents the projection of dataset X onto new axis system (the weights matrix). $W = X_m * P_k = (X - m) * P_k$

P_k is a MN x k matrix, that consists of k eigenfaces, thus the resulting matrix will be n x k, where n is the number of images in dataset.

Now, when we want to identify the person on a photo, we need to first project the image onto new principal axis: $Img = (I - m) * P_k$, where I is an image we try to classify.

The result will be a 1xK vector. We then need to find the best match to the given projected image in the dataset. To do so we find the least distance to all of the projected photos in our dataset (to every row in W matrix).

$$L(Img, Iw) = \sum_{i=1}^{k} |Img_i - Iw_i|$$

The nearest fit (the smallest distance) will indicate the matching person identity to given image. Face detection is done.

Implementation pipeline

At this moment, we have implemented the PCA method for face recognition (the following material/code will be posted on github soon). We have used face database provided by Yale University. In the nearest future we plan to add photos of us and students at UCU, so the implementation could identify them. We also plan wo create appealing interface for our implementation and, if possible, add the function of face detection, so you could feed the program the full-length photo of a person.