Face recognition and Image compression using SVD and PCA

Team: “Basic “
Ahmed Abbas Sayed 1711018 - Gad Mohamed 1710341
Mohamed Hassan 1710710 – Rawan Abdullah 1710570

NILE UNIVERSITY | MATH-301 LINEAR ALGEBRA | DR. MOHAMED EL-BELTAGY

Contents

Abstract- Human beings can detect and recognize faces skillfully even when conditions like face expression, lighting condition, or viewing angle change. In this paper, we describe a procedure for facial recognition using PCA and SVD after explaining both concepts thoroughly. To validate our theoretical results, we develop a facial recognition application using linear dimensionality reduction techniques. Also, we'll spot light on the extreme image compression potential of these techniques.

Literature review

In the AI field, scalability is a rising problem, algorithms work well on small data but when tested on big data sets, they're extremely slow, our application is no exception. Since we are dealing with images, which are often stored as m x n matrix. So, for example, if we have 1000 image in the training set, each of dimensions 120 * 100, that's about 12 million pixels, and computations on this huge data will be quite heavy. Hence, a usual practice in reinforced learning algorithms is to use dimensionality reduction techniques.

These are generally of two types: [1]

1- Feature selection: where we choose what we think are the most important dimensions of the data and abandon other dimensions.
2- Feature extraction: where we use linear algebra tools (like "singular value decomposition" SVD) to "extract" the most important dimensions by combining two or more original dimensions to form some new perspective of data that shows most of its features without having to deal with unimportant dimensions.

To detect faces you can' t just ignores "unimportant" features of face, but we can make "combinations" of the most important features of all faces (i.e. eigen faces) then we can represent any face as a combination of weighted eigen faces.

In what follows, we explain more in details the concepts of "principle component analysis" PCA and "singular value decomposition" SVD.

Singular –Value Decomposition

The matrix decomposition which is also known as matrix factorization can be described by using its constituent elements.

The Singular-Value Decomposition (SVD) is the most known and widely used matrix decomposition, as it is known that all matrices have SVD, so it makes it more stable and profitable than any other method.

SVD can be used in different application and fields such as Eigen decomposition, it can also be used in wide array of applications including compressing, denoising, and data reduction.

So, what is the Singular –Value Decomposition (SVD)?

SVD is matrix decomposition method for reducing the matrix to its constituent parts to make certain subsequent matrix calculations simple [1]

The SVD can be calculated through different numerical methods.

There is singular –value decomposition for every rectangular matrix, although in some cases the resulting matrix may contain complex numbers and the limitation of floating-point arithmetic may cause some matrices to fail to decompose neatly.

By using the method of (SVD), it has made it easy to find another way to factories the matrix into singular matrix and singular values.

The singular –value decomposition help in discovering some of the information as the eigen

decomposition, although the (SVD) is more generally applicable.

The SVD can be used in the calculation of other matrix operation such as matrix inverse, it can also be used as a date reduction method in machine learning. [6]

Now, it has become obvious that, SVD can be used in least squares linear regression, denoising data, and image compression.

The singular value decomposition has affected widely in some fields and application as it has a numerous application in machine learning, statics, and computer science.

The formula of SVD is as follows: [6]

$$\boldsymbol{A = U\Sigma V^T} \qquad \textbf{(1)}$$

, where A is an m*n data matrix, U is also an m*n matrix with orthonormal columns (i.e. forms the vector basis or the eigen values of its space), Σ is a diagonal matrix (i.e. all off-diagonal entries are zero) represents the singular values which is directly related to the eigen values of U's columns, and V^T is an n*n matrix.

Fig.1 shows a graphical representation of SVD:

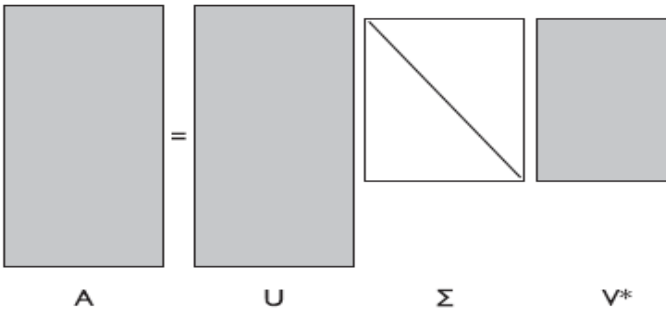

Fig.1

Principle component analysis

Multivariate Analysis often starts out with data involving a substantial number of correlated variables.

Principal Component Analysis (PCA)is a dimension-reduction tool that can be used to reduce a large set of variables to a small set that still contains most of the information in the large set.

Principle component analysis (PCA) is a very important method for dimensionally reduction, which uses simple matrix operation from linear algebra and statics to calculate a projection of the original data into the same number or fewer dimensions.

Principle component analysis can be known as a projection method where data of M columns can be projected into a subspace with m or fewer m columns.

The method of principle component (PLC) can be implemented using linear algebra tools.

Principal component analysis (PCA) is a mathematical procedure that transforms several (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components.

The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

Traditionally, principal component analysis is performed on a square symmetric matrix.

A correlation matrix is used if the variances of individual varies differ much, or if the units of measurement of the individual varieties differ.

Fig.2 shows data points being transformed from the x-y-z dimensional system to another principle basis that highlight more the variance of data and allow us to reduce the number of

dimensions without losing much of data meaning.

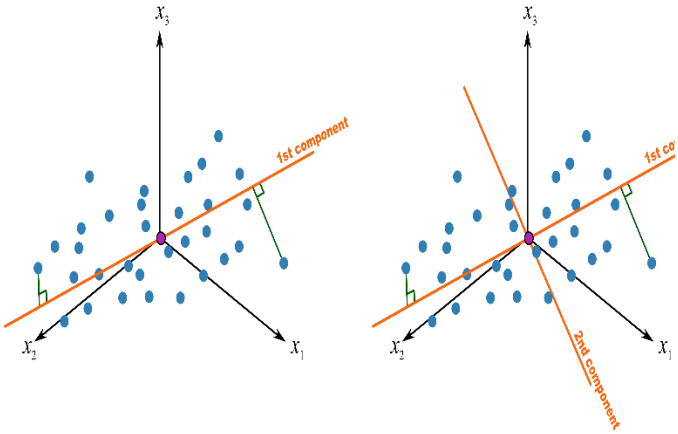

Fig.2

Calculating PCA:

PCA chooses the best principle component (L) that represents datapoints. This is done using one of two methods:

1- Reducing squared-distance-error e, shown on Fig. 3

$$\mathbf{J(L)} = \sum_{\mathbf{k=i}}^{\mathbf{m}} \|(\mathbf{X_k} - \mu) - \mathbf{L^t}(\mathbf{X_k} - \mu)\mathbf{L}\|^2 \quad \textbf{(3)}$$

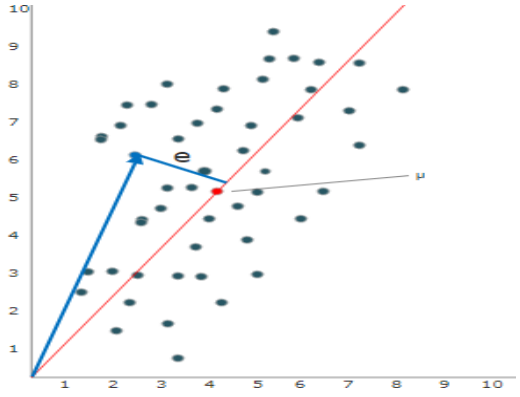

Fig. 3

2- Maximize the variance projection of datapoints, shown on Fig.2

$$\mathbf{R(L)} = \sum_{\mathbf{k=1}}^{\mathbf{m}} \frac{\mathbf{1}}{\mathbf{N}} \|\mathbf{L^t}(\mathbf{X_k} - \mu)\|^2 \quad \textbf{(4)}$$

Where, $\mathbf{X} = \mu + (\mathbf{X} - \mu)$

Note that X doesn't refer to the x-axis but to vector representing the point location, whether in 3d space or other spaces.

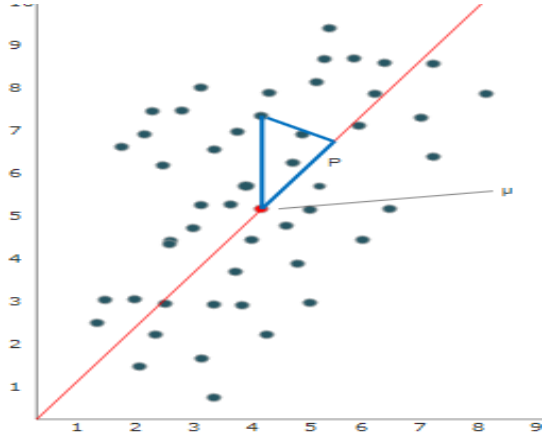

Fig. 4

Both methods lead to one goal. Note that since we have a constraint $\mathbf{L^T L = 1}$, this is considered as an optimization problem using LaGrange multiplier.

Also, note that if we simplified R(L) to

$$\frac{\mathbf{1}}{\mathbf{N}}\mathbf{L^t}\ (\sum_{\mathbf{k=1}}^{\mathbf{m}}(\mathbf{X_k} - \mu)(\mathbf{X_k} - \mu)^{\mathbf{t}})\mathbf{L} \quad \textbf{(5)}$$

we'll find that the term $(\mathbf{X_k} - \mu)(\mathbf{X_k} - \mu)^{\mathbf{t}}$is the covariance matrix thus,

$$\mathbf{R(L)= L^t \sum L} \quad \textbf{(6)}$$

$$\mathbf{g(L) = L^t L - 1} \quad \textbf{(7)}$$

$$\mathbf{R_2(L) = L^t \sum L - \lambda\, g(L)}$$
$$\mathbf{= L^t \sum L - \lambda(L^t L - 1)} \quad \textbf{(8)}$$

Differentiation:

$$\frac{\mathbf{R_2(L)}}{\mathbf{L}} = \mathbf{2 \sum L - \lambda(2L)} \quad \textbf{(9)}$$

$$\mathbf{2 \sum L = 2\lambda L} \quad \mathbf{, \sum L = \lambda L} \quad \textbf{(10)}$$

Next, we diagonalize the covariance matrix to its eigen values and eigen vectors. Since any covariance matrix is symmetric, it has the following features:

1- Its eigenvectors are orthogonal
2- $P^{-1} = P^{T}$

Thus,

$$\boldsymbol{variance = L^{t}\Sigma\, D\, \Sigma^{T} L} \qquad \textbf{(11)}$$

The variance equations show that the variance of any eigenvector is itself the eigenvalue of that same eigenvector. So, if we want the principle component that shows the most details of the data, we choose the eigenvector corresponding to the highest eigenvector and so on.

Relating PCA and SVD

Singular value decomposition (SVD) and principal component analysis (PCA) are two eigenvalue methods used to reduce a high-dimensional dataset into fewer dimensions while retaining important information. [5]

PCA viewpoint requires that one compute the eigenvalues and eigenvectors of the covariance matrix, which is the product XXTXXT, where XX is the data matrix.

But since finding the covariance usually results in loss of data precision (more on this in the methodology section) we use SVD which is directly related to PCA but, unlike PCA, SVD doesn't require the covariance matrix to be calculated. [5] [6]

The relation between SVD and PCA can be derived as follows:

1- Since the covariance matrix is symmetric, the matrix is diagonalizable, and the eigenvectors can be normalized such that they are orthonormal:

$$\boldsymbol{AA^{T} = A\mathrm{D}\, A^{T}} \qquad \textbf{(12)}$$

, where $\boldsymbol{A}$ is the data matrix and $\boldsymbol{AA^{T}}$ is its covariance matrix.

2- Now, applying SVD to the data matrix $\boldsymbol{A}$ as follows:

$$\boldsymbol{A = U\Sigma V^{T}} \qquad \textbf{(13)}$$

3- Multiplying both sides with $\boldsymbol{A^{T}}$ yields:

$$\boldsymbol{AA^{T} = (U\Sigma V^{T})(U\Sigma V^{T})\mathrm{T}} \qquad \textbf{(14)}$$

$$\boldsymbol{AA^{T} = U\Sigma V^{T})(V\Sigma U^{T})} \qquad \textbf{(15)}$$

4- Since **V** is an orthogonal matrix:

$$\boldsymbol{VV^{T} = 1} \qquad \textbf{(16)}$$

It's apparent from the previous derivation that SVD and PCA are closely related and that we use SVD to avoid having to calculate the covariance which result in precision loss of our data.

Methodology

We'll start our work by gathering a training set of face pictures. This set must be collected carefully to have one position of the face (i.e. one viewpoint of faces in all pictures with eyes aligned) to successfully align the dimensions in the matrix of our dataset. In the figure below, we show the difference between a picture that will form meaningful mean face and one that won't. [1]

Fig.5

Two simulations will be delivered:

1- The actual face recognition simulation where the target face is stored in a dataset among other pictures and when the target appears in front of the camera the program will be show recognize him and display his image from the dataset.
2- A simulator that shows the potential to use this technique for compression purposes. This will be done by building two applications, a sender and a receiver, where we attempt to send a big images dataset using PCA and compare it with the normal file sharing in terms of time.

Building a Dataset:

We've used two datasets: one that will be used in the face recognition simulator and it contains aligned frontal face images of some NU students. The sample looks like the following:

And the other dataset is > 1000 images from CelebA (a free dataset on the internet) and we use it to prove the concept and show clearly the result mean face.

All images are resized to 150x150x3. The images show different facial expressions and lighting conditions which we found very important in accurate recognition.

What we want to achieve is to change the coordinate system from its standard form to another more informative one. So that the origin will not be the (0, 0, 0, …, 0) point but will be the mean face and the axes will not be the standard basis but will be the Principle Components which are the eigen vectors of the covariance matrix of our data matrix. [1]

steps

1- The first step is to build our data matrix A by flattening all images in our dataset and stack them row on row. For example, if we have 150 images and each of them is of dimension 150x150x3. We convert this data set to a 150x67500 matrix.
2- We calculate the mean face (our new reference point). note that the mean face created is still flattened and we will have to reshape it from 1x67000 vector to a 150x150x3 matrix. The resulting mean face:

Fig.6

3- We subtract the mean from each row.
4- We obtain A's principle components using any of dimensionality reduction tools (PCA or SVD), we calculate SVD like [4]:

Important note: in the code above, you can see that although the we need to calculate the PCA, we exchanged the third step with SVD. This is because using the SVD to perform PCA makes much better sense numerically than forming the covariance matrix to begin with, since the formation of AA^T can cause loss of precision. This can be seen in the figure below which represents the Läuchli matrix which can be stable SVD'd but forming AA^T can be disastrous. [5] [6]

$$\begin{pmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{pmatrix}^T$$

where ϵ is a tiny number.

Fig. 7

SVD decomposes the face matrix into three parts, **USV**, where **U** and **V** are the left and right singular vectors of **A**, respectively, and **S** is a diagonal matrix whose elements are the singular values of **A**.
the columns of **U** form the eigen vectors of the eigenspace of **A** 's covariance matrix, each column corresponds to an eigenface!

So, what does the eigen values shows? It shows the most common features among a group of face pictures (i.e. the correlation between pixels). Hence our goal of recognizing faces becomes, simply, looking for correlations between eigenfaces. Two pictures that're highly correlated are likely to be for the same person.

Also note that a property of the eigenvalues produced from PCA or singular values produced from SVD is that they are arranged from biggest (corresponds to the most important eigenface) to smallest number (corresponds to the least important eigenface).

Results

Face recognition

Fig. 8 shows the projection of 3 images in the dataset on the new eigenvectors

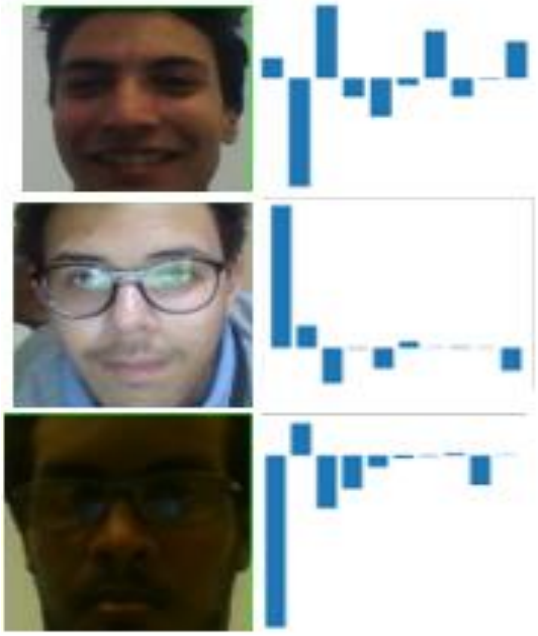

Fig. 8

A test image (Fig. 9) from outside the dataset is used. Firstly, we will get its projection on the eigenvectors.

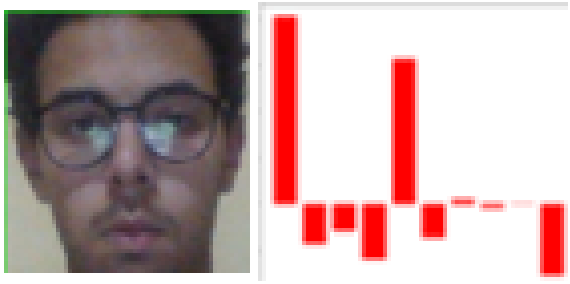

Fig. 9

Finally, the distance between the weights of the test image and the weights of the dataset images is calculated and the closest dataset image to the test image is chosen as the result

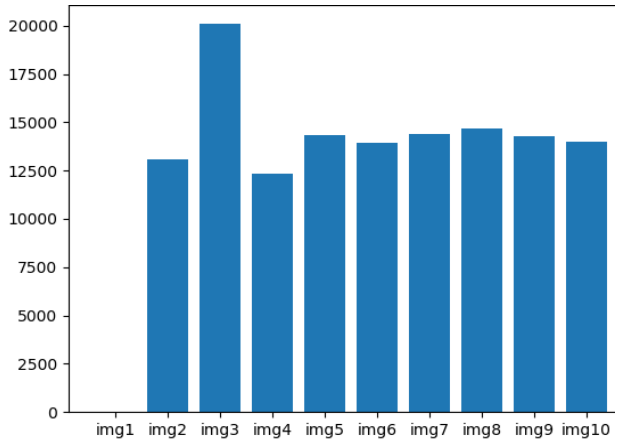

Fig. 10

The distance with the first image is zero because it's itself the test image, it was added to the dataset before making this plot to clarify the meaning of the distance between images.

The lowest 3 distances are with img2, img4, and img6. But two of them (img2 and img6) belong to a different person i.e. are wrong answers. The right images are img4, img8, and img8. And since img4 is the global minimum distance, the recognition of the face in the test image will be correct.

the next image shows a screenshot from the life simulation where a person is first detected through camera using OpenCV library, a green square is drawn around his face, the face is then processed using PCA against the dataset, and the result is popped up in another window as shown.

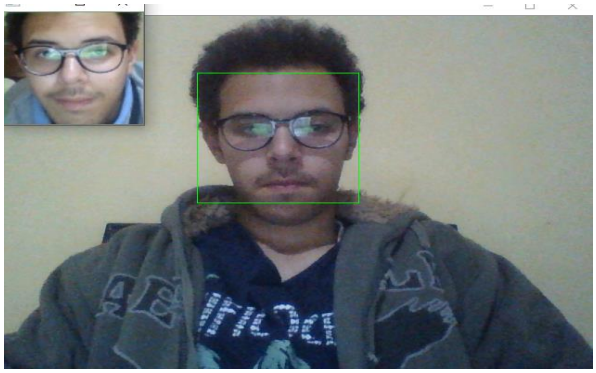

Fig. 11

Image compression:

This part was added to this project to make use of the extreme compression capabilities of PCA. Instead of saving the original dimensions of and image (which are 200x200x3= 120000 dimension for a 200x200 image) we calculate its projection on the eigenvectors and save the weights on the eigenvectors instead.

The image can be reconstructed using:

$$\boldsymbol{image = mean + \sum_{k=1}^{m} V_k W_k} \qquad \textbf{(17)}$$

Where, V is an eigenvector and W is the weights of the image on that eigenvector.

The compression factor is 83:4000. The weight stored are floats not bytes like the 120000 weights of the traditional image. For example, the weight of the image on the 1st eigenvector might be something like 28.32498132. the compression factor can be further improved using integer or byte values instead of float numbers to reach 11: 2000 or 0.005.

But, how is this possible? For 2 reasons:

1- The eigenvectors are positioned in a place relative to the data that make it much more capable of representing the deviations in them with only few numbers of dimensions. i.e. it's like the eigenvector location is handcrafted to represent specific static data using the lowest possible number of dimensions.

2- The huge space cutoff is added to the processing needed. Since this image has its own reference point and its own basis, processing is needed to transform the image from the traditional reference point and basis vectors to our handcrafted ones.

Results discussion

To understand the process of face recognition, consider the following picture:

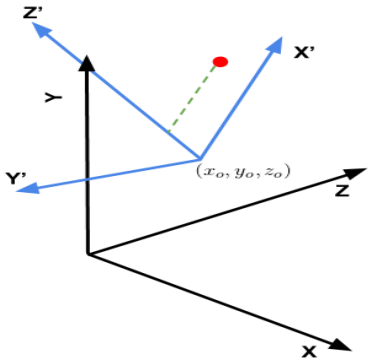

Fig. 12

This is an analogy to understand how faces are reconstructed and compared with the dataset.

Let's assume that the red point is an image in a 3-dimensional space represented by the points, or weights, (x, y, z) with respect to the natural basis XYZ and the original point is (0, 0, 0). Now we want to know the weights of this red point with respect to another origin (x0, y0, z0), and another basis $X'Y'Z'$ which corresponds to the mean face and the eigen vectors in our case.

The question is, what are the weights of the red dots on each of the basis of the new coordinate system? This can be found by two steps:

1- Translation: we remove the translation component from (x, y, z) by subtracting the new origin (x0, y0, z0), the result is a new vector (x- x0, y- y0, z- z0). (therefore, we subtracted each image of our dataset from the mean face)

2- Projection: we project the new vector (x- x0, y- y0, z- z0) on the new basis $X'Y'Z'$ which is dot product the vector with X' and dot product the vector with Y' and dot product the vector with Z'.

We need to perform these two steps on each image we have in the dataset, given that the new basis is the eigen vectors of the covariance matrix of our data matrix. The new basis is more informative because they are orthogonal to the most variation of the data.

Conclusion

In this paper, we've used PCA in two ways: as a feature extraction tool and as a dimensionality reduction tool.

1. Feature extraction for face recognition
2. Dimensionality reduction for image compression

Based on the results, the distance between the test image and the dataset images shows that using dimensionality reduction tools has its limitations like: it's not robust to misalignment or changing lighting conditions.
Also, image compression works only on images that're part of the dataset from which we calculated the mean and eigenvectors.

References:

[1] Kutz, J. (2013). *Data-driven modeling and scientific computation*. Oxford [etc.]: Oxford U.P.

[2] Turk, M. and Pentland, A. (1991). Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1), pp.71-86.

[3] Learnche.org. (2019). *6.5.2. Geometric explanation of PCA — Process Improvement using Data*. [online] Available at: https://learnche.org/pid/latent-variable-modelling/principal-component-analysis/geometric-explanation-of-pca [Accessed 31 Mar. 2019].

[4] Brownlee, J. (2019). *How to Calculate Principal Component Analysis (PCA) from Scratch in Python*. [online] Machine Learning Mastery

[5] Learnche.org. (2019). *6.5.2. Geometric explanation of PCA — Process Improvement using Data*. [online] Available at: https://learnche.org/pid/latent-variable-modelling/principal-component-analysis/geometric-explanation-of-pca [Accessed 31 Mar. 2019].

[6] Brownlee, J. (2019). *A Gentle Introduction to Singular-Value Decomposition for Machine Learning*. [online] Machine Learning Master