REPORT ON
PRINCIPAL COMPONENT ANALYSIS (PCA)

SUBMITTED BY:

Shravya Suresh,

3rd year, B.Tech,

Computer Science Engineering,

Bangalore Institute of Technology,

Engineering, Bengaluru

GUIDED BY:

I Visveswaran,

DGM,Architect,

Tata Power SED,

Bengaluru

CERTIFICATE

This is to certify that the following documentation of the '**Principal Component Analysis'** have been compiled and constructed by **Shravya Suresh**,3rd year Computer Science Engineering student at **Bangalore Institute of Technology**, Bengaluru. It contains the outcome of learning, experience and interaction of the application I learned during the Internship at **TATA POWER SED,BENGALURU.**

Guide:

BAMA J
TATA POWER SED,
BENGALURU

Head:

I VISVESWARAN
DGM,ARCHITECT
TATA POWER SED

BENGALURU

ACKNOWLEDGEMENT

The sincere support and guidance of the entire organization has resulted in this project.

Firstly I would like to render my deepest gratitude to I Visveswaran,DGM(Architect) Sir for his very helpful mentoring on Machine Learning and Deep Learning. I would like to thank him for taking time off his busy hours to help me with the entire learning and implementation.I would also like to thank him for all the important industry and personality development tips.

I would also like to thank Bama J Ma'am for extending her support in the python programming and also guiding me throughout the documentation.

Lastly, I am grateful to all my fellow colleagues for being ever supportive and cooperating.

CONTENTS

1.Introduction……5

2.Working of PCA……8

3.Properties of PCA……..11

4.Implementation of PCA on a 2D set……..12

5.Application of PCA……15

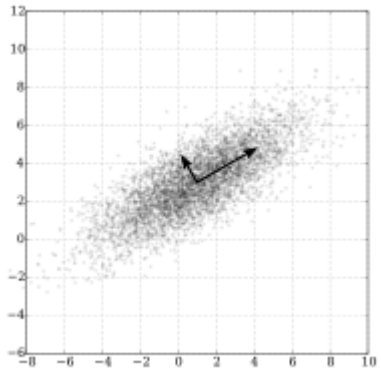
6.Python implementation of PCA……..18

7.Conclusion……..28

PRINCIPAL COMPONENT ANALYSIS

INTRODUCTION:

Principal component analysis (**PCA**) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called **principal components**. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing *n* observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.



PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. PCA can be done by eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition of a data matrix, usually after a normalization step of the initial data. The normalization of each attribute consists of *mean centering* – subtracting each data value from its variable's measured mean so that its empirical mean (average) is zero – and, possibly, normalizing each variable's variance to make it equal to 1; see Z-scores.[4] The results of a PCA are usually discussed in terms of *component scores*, sometimes called *factor scores* (the transformed variable values corresponding to a particular data point), and *loadings* (the weight by which each standardized original variable should be multiplied to get the component score).[5] If component scores are standardized to unit variance, loadings must contain the data variance in them (and that is the magnitude of eigenvalues). If component scores are not standardized (therefore they contain the data variance) then loadings must be unit-scaled, ("normalized") and these weights are called eigenvectors; they are the cosines of orthogonal rotation of variables into principal components or back.

PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multivariate dataset is visualised as a set of coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture, a projection of this object when viewed from its most informative viewpoint[*citation needed*]. This is done by

using only the first few principal components so that the dimensionality of the transformed data is reduced.

PCA is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix.

PCA is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset.

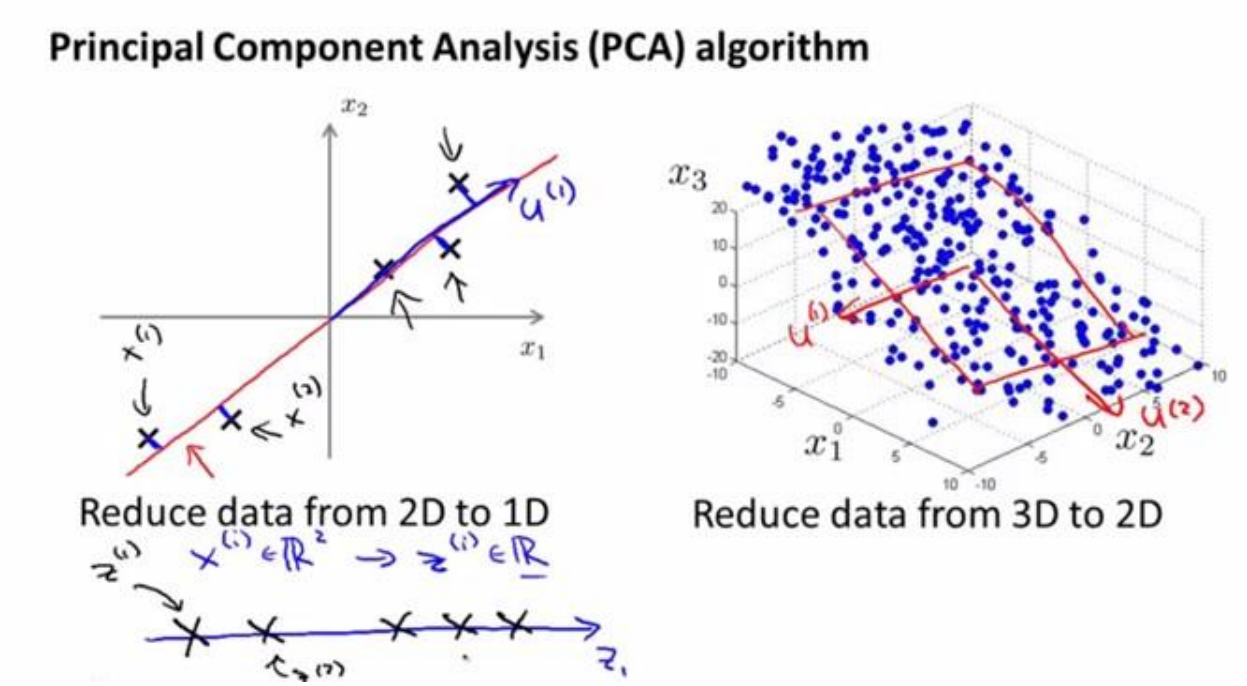
Robust and L1-norm-based variants of standard PCA have also been proposed.

WORKING OF PCA:

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the 1st principal component retains maximum variation that was present in the original components. The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.

Importantly, the dataset on which PCA technique is to be used must be scaled. The results are also sensitive to the relative scaling. As a layman, it is a method of summarizing data. Imagine some wine bottles on a dining table. Each wine is described by its attributes like colour, strength, age, etc. But redundancy will arise because many of them will measure related properties. So what PCA will do in this case is summarize each wine in the stock with less characteristics.

Intuitively, Principal Component Analysis can supply the user with a lower-dimensional picture, a projection or "shadow" of this object when viewed from its most informative viewpoint.



- Dimensionality : It is the number of random variables in a dataset or simply the number of features, or rather more simply, the number of columns present in your dataset.

- Correlation : It shows how strongly two variable are related to each other. The value of the same ranges for -1 to +1. Positive indicates that when one variable increases, the other increases as well, while negative indicates the other decreases on increasing the former. And the modulus value of indicates the strength of relation.

- Orthogonal: Uncorrelated to each other, i.e., correlation between any pair of variables is 0.

- Eigenvectors: Eigenvectors and Eigenvalues are in itself a big domain, let's restrict ourselves to the knowledge of the same which we would require here. So, consider a non-zero vector ***v***. It is an eigenvector of a square matrix ***A***, if ***Av*** is a scalar multiple of **v**. Or simply:
  $\boldsymbol{Av = \lambda v}$

  Here, ***v*** is the eigenvector and $\boldsymbol{\lambda}$ is the eigenvalue associated with it.

- Covariance Matrix: This matrix consists of the covariances between the pairs of variables. The *(i,j)*th element is the covariance between *i*-th and *j*-th variable.

PROPERTIES OF PRINCIPAL COMPONENT:

Technically, a principal component can be defined as a linear combination of optimally-weighted observed variables. The output of PCA are these principal components, the number of which is less than or equal to the number of original variables. Less, in case when we wish to discard or reduce the dimensions in our dataset. The PCs possess some useful properties which are listed below:

1. The PCs are essentially the linear combinations of the original variables, the weights vector in this combination is actually the eigenvector found which in turn satisfies the principle of least squares.
2. The PCs are orthogonal, as already discussed.
3. The variation present in the PCs decrease as we move from the 1st PC to the last one, hence the importance.

   The least important PCs are also sometimes useful in regression, outlier detection, etc.

Pre-solved code recipes usually help in finishing your projects faster.

IMPLEMENTING PCA ON A 2D SET:

Step 1: Normalize the data

First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become ***x***- and all Y become ***y***-. This produces a dataset whose mean is zero.

Step 2: Calculate the covariance matrix

Since the dataset we took is 2-dimensional, this will result in a 2x2 Covariance matrix.

$$Matrix(Covariance) = \begin{bmatrix} Var[X_1] & Cov[X_1, X_2] \\ Cov[X_2, X_1] & Var[X_2] \end{bmatrix}$$

Please note that $Var[X_1] = Cov[X_1,X_1]$ and $Var[X_2] = Cov[X_2,X_2]$.

Step 3: Calculate the eigenvalues and eigenvectors

Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix. The same is possible because it is a square matrix. λ is an eigenvalue for a matrix ***A*** if it is a solution of the characteristic equation:

$$det(\lambda I - A) = 0$$

Where, ***I*** is the identity matrix of the same dimension as ***A*** which is a required condition for the matrix subtraction as well in this case and '***det***' is the determinant of the matrix. For each eigenvalue λ, a corresponding eigen-vector ***v***, can be found by solving:

$$(\lambda I - A)v = 0$$

Step 4: Choosing components and forming a feature vector:

We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with *n* variables, then we have the corresponding *n* eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first *p* eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.

Next we form a feature vector which is a matrix of vectors, in our case, the eigenvectors. In fact, only those eigenvectors which we want to proceed with. Since we just have 2 dimensions in the running example, we can either choose the one corresponding to the greater eigenvalue or simply take both.

$$Feature\ Vector = (eig_1, eig_2)$$

Step 5: Forming Principal Components:

This is the final step where we actually form the principal components using all the math we did till here. For the same, we take the transpose of the feature vector and left-multiply it with the transpose of scaled version of original dataset.

$$NewData = FeatureVector^T \times ScaledData^T$$

Here,

NewData is the Matrix consisting of the principal components,

FeatureVector is the matrix we formed using the eigenvectors we chose to keep, and
ScaledData is the scaled version of original dataset
('T' in the superscript denotes transpose of a matrix which is formed by interchanging the rows to columns and vice versa. In particular, a 2x3 matrix has a transpose of size 3x2)

If we go back to the theory of eigenvalues and eigenvectors, we see that, essentially, eigenvectors provide us with information about the patterns in the data. In particular, in the running example of 2-D set, if we plot the eigenvectors on the scatterplot of data, we find that the principal eigenvector (corresponding to the largest eigenvalue) actually fits well with the data. The other one, being perpendicular to it, does not carry much information and hence, we are at not much loss when deprecating it, hence reducing the dimension.

All the eigenvectors of a matrix are perpendicular to each other. So, in PCA, what we do is represent or transform the original dataset using these orthogonal (perpendicular) eigenvectors instead of representing on normal ***x*** and ***y*** axes. We have now classified our data points as a combination of contributions from both ***x*** and ***y.*** The difference lies when we actually disregard one or many eigenvectors, hence, reducing the dimension of the dataset. Otherwise, in case, we take all the eigenvectors in account, we are just transforming the co-ordinates and hence, not serving the purpose.

APPLICATIONS OF PCA:

PCA is predominantly used as a dimensionality reduction technique in domains like facial recognition, computer vision and image compression. It is also used for finding patterns in data of high dimension in the field of finance, data mining, bioinformatics, psychology, etc.

Pre-solved code recipes usually help in finishing your projects faster.

PCA for images:

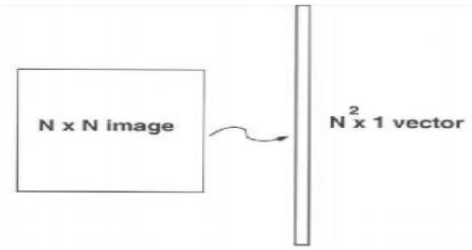
You must be wondering many a times show can a machine read images or do some calculations using just images and no numbers. We will try to answer a part of that now. For simplicity, we will be restricting our discussion to square images only. Any square image of size ***NxN*** pixels can be represented as a ***NxN*** matrix where each element is the intensity value of the image. (The image is formed placing the rows of pixels one after the other to form one single image.) So if you have a set of images, we can form a matrix out of these matrices, considering a row of pixels as a vector, we are ready to start principal component analysis on it. How is it useful ?

Say you are given an image to recognize which is not a part of the previous set. The machine checks the differences between the to-be-recognized image and each of the principal components. It turns out that the process performs well if PCA is applied and the differences are taken from the 'transformed' matrix. Also, applying PCA gives us the liberty

to leave out some of the components without losing out much information and thus reducing the complexity of the problem.

For image compression, on taking out less significant eigenvectors, we can actually decrease the size of the image for storage. But to mention, on reproducing the original image from this will lose out some information for obvious reasons.

PYTHON IMPLEMENTATION OF PCA:



- **Main idea behind eigenfaces**

- Suppose Γ is an N^2x1 vector, corresponding to an NxN face image I.

- The idea is to represent Γ ($\Phi = \Gamma$ - mean face) into a low-dimensional space:

$$\hat{\Phi} - mean = w_1 u_1 + w_2 u_2 + \cdots w_K u_K \ (K << N^2)$$

Computation of the eigenfaces

Step 1: obtain face images $I_1, I_2, ..., I_M$ (training faces)

(**very important:** the face images must be *centered* and of the same *size*)

Step 2: represent every image I_i as a vector Γ_i

Step 3: compute the average face vector Ψ: $\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$

Step 4: subtract the mean face: $\Phi_i = \Gamma_i - \Psi$

Step 5: compute the covariance matrix C:

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T \quad (N^2 \text{x} N^2 \text{ matrix})$$

where $A = [\Phi_1 \ \Phi_2 \cdots \Phi_M] \quad (N^2 \text{x} M \text{ matrix})$

Step 6: compute the eigenvectors u_i of AA^T

The matrix AA^T is very large --> not practical !! (N^2xN^2 matrix)

Step 6.1: consider the matrix $A^T A$ (MxM matrix)

Step 6.2: compute the eigenvectors v_i of $A^T A$: $A^T A v_i = \mu_i v_i$

AA^T and $A^T A$ have the same eigenvalues and their eigenvectors are related as follows: $u_i = A v_i$!!

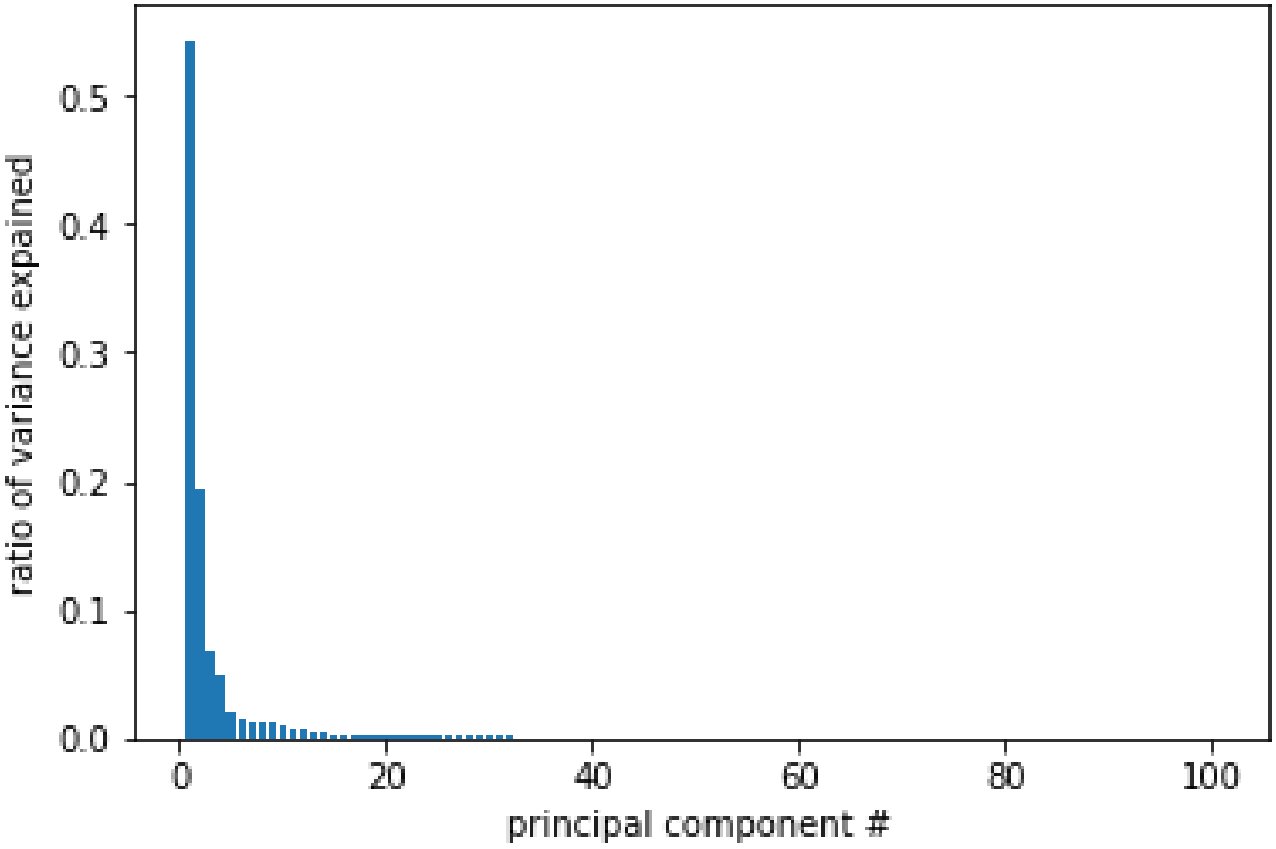
Step 6.3: compute the M best eigenvectors of AA^T: $u_i = A v_i$

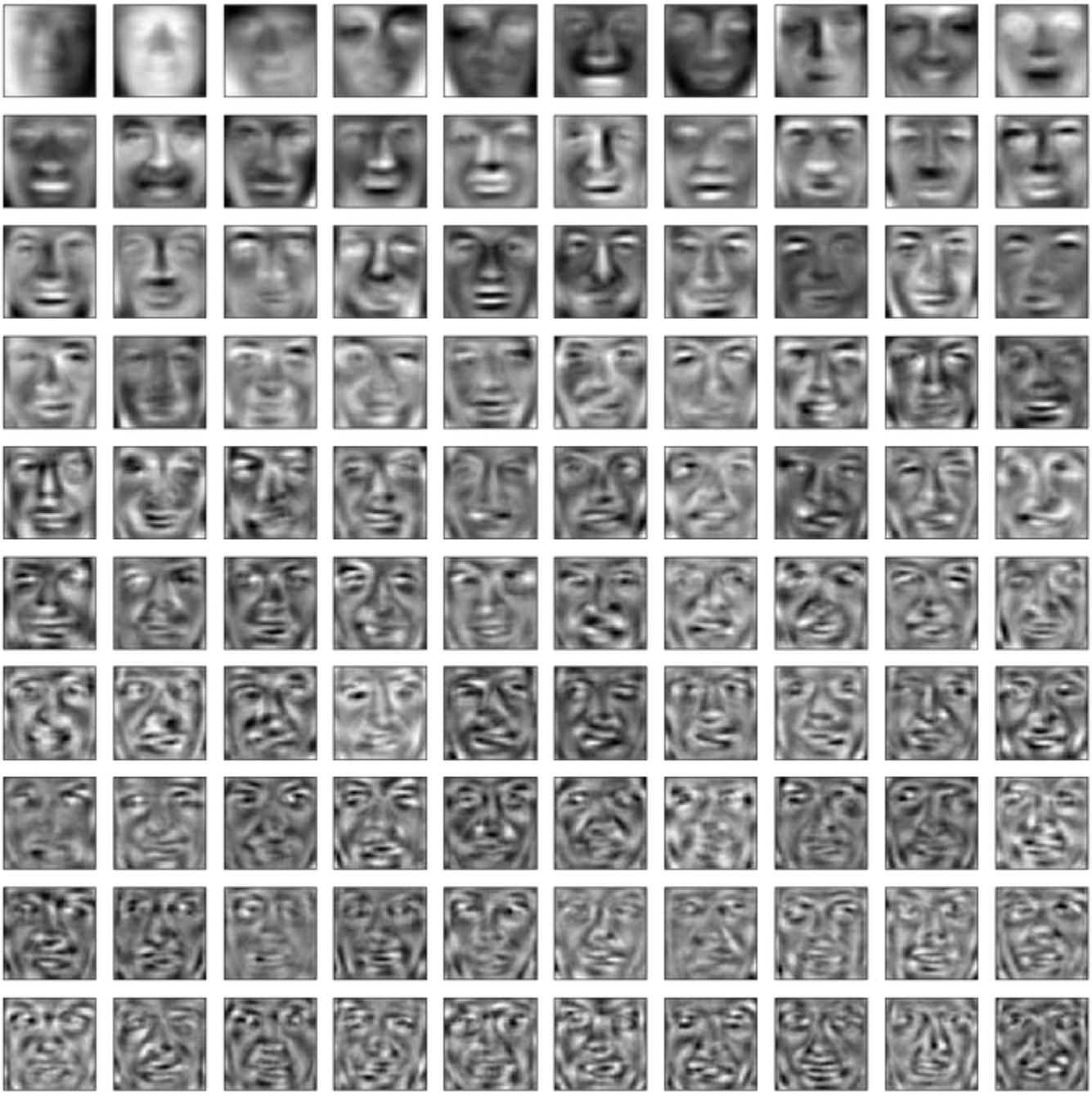
(**important:** normalize u_i such that $\|u_i\| = 1$)

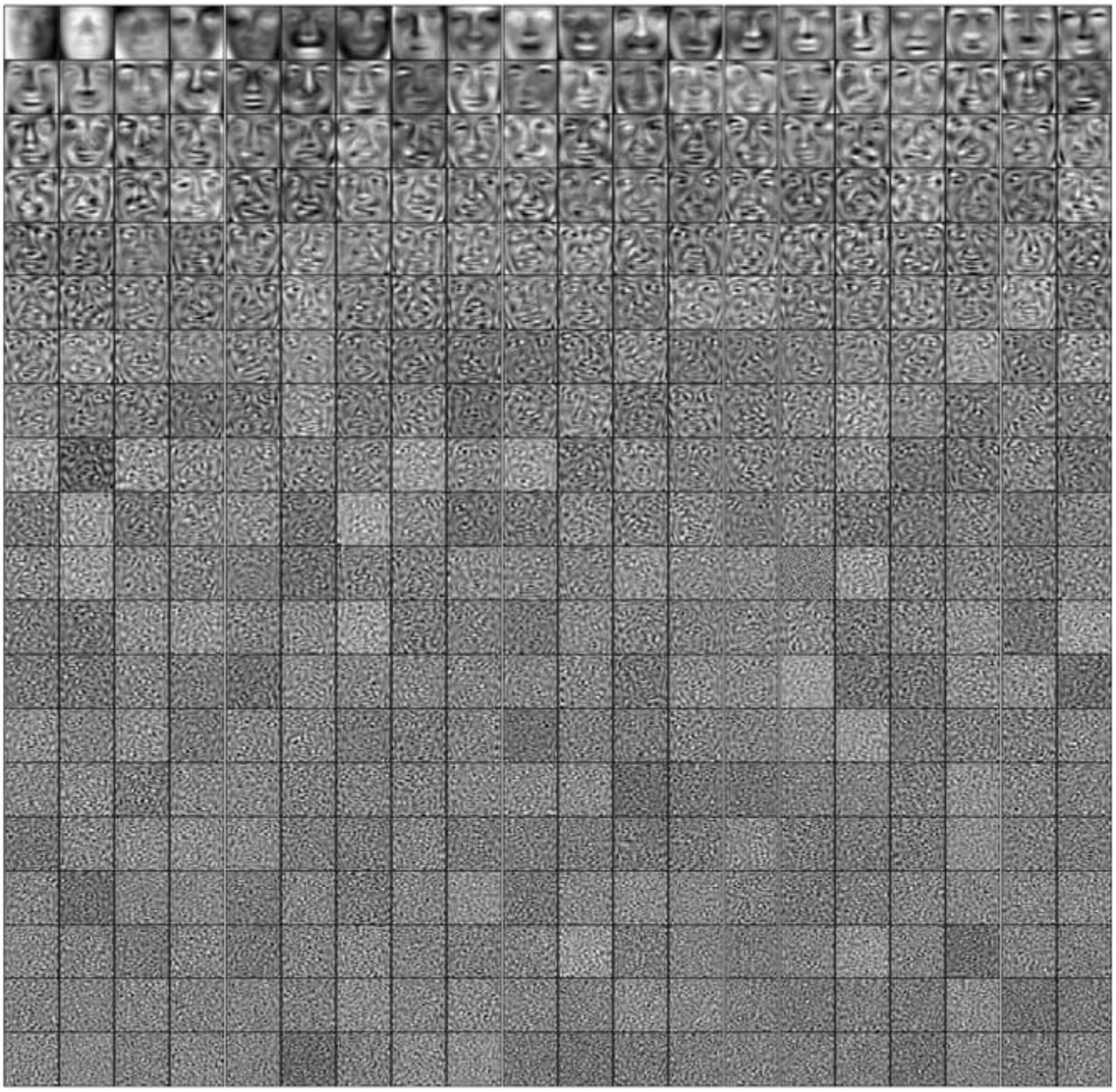
Step 7: keep only K eigenvectors (corresponding to the K largest eigenvalues)

Taken from the case study Eigenfaces for Face Detection/Recognition,
http://www.vision.jhu.edu/teaching/vision08/Handouts/case_study_pca1.pdf

The next 4 figures show the **meanface**, the percentage **variance** captured by the eigenfaces and the **top 100** and then **top 400 eigenfaces** computed computed from the images, respectively.







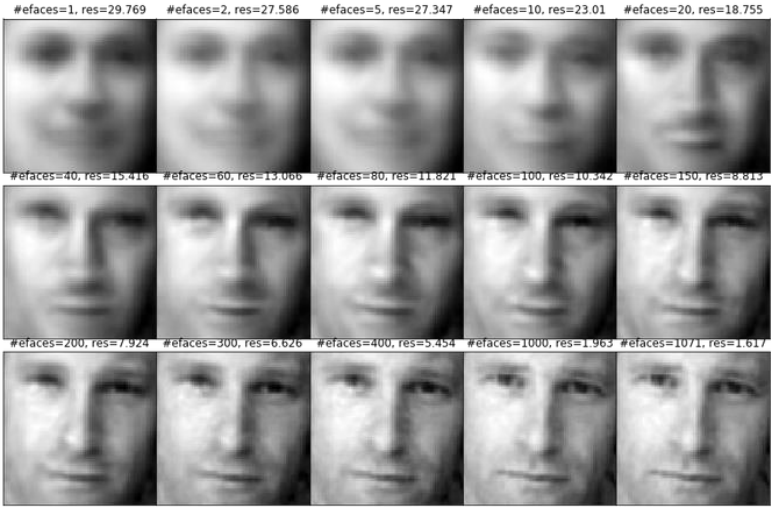
Projection on the EigenFaces and Reconstruction

Now, let's project a face onto the face space to generate a vector of k coefficients, one for each of the **k eigenfaces** (for different values of **k**). Then let's reconstruct the same face from the vector of coefficients computed.
The following figures and animations show how a given image can be reconstructed by projecting on the first few dominant eigenfaces and also how the reconstruction error (**residual**) decreases as more and more eigenfaces are used for projection.

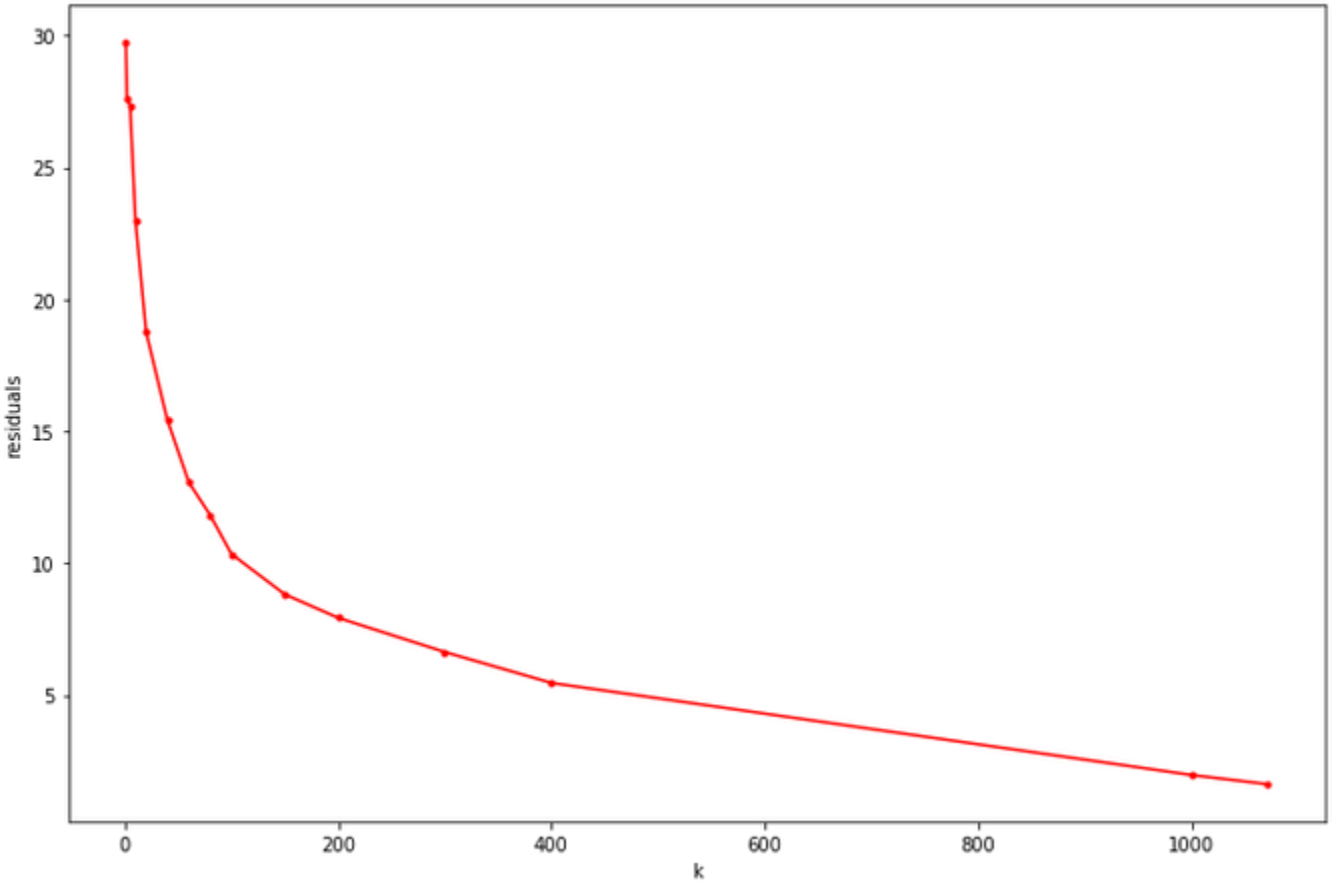
Also, the reconstruction error goes down quite fast, with the selection of first few eigenfaces.

Original Face Image

Reconstructed Face Image by projecting on the truncated eigenfaces space

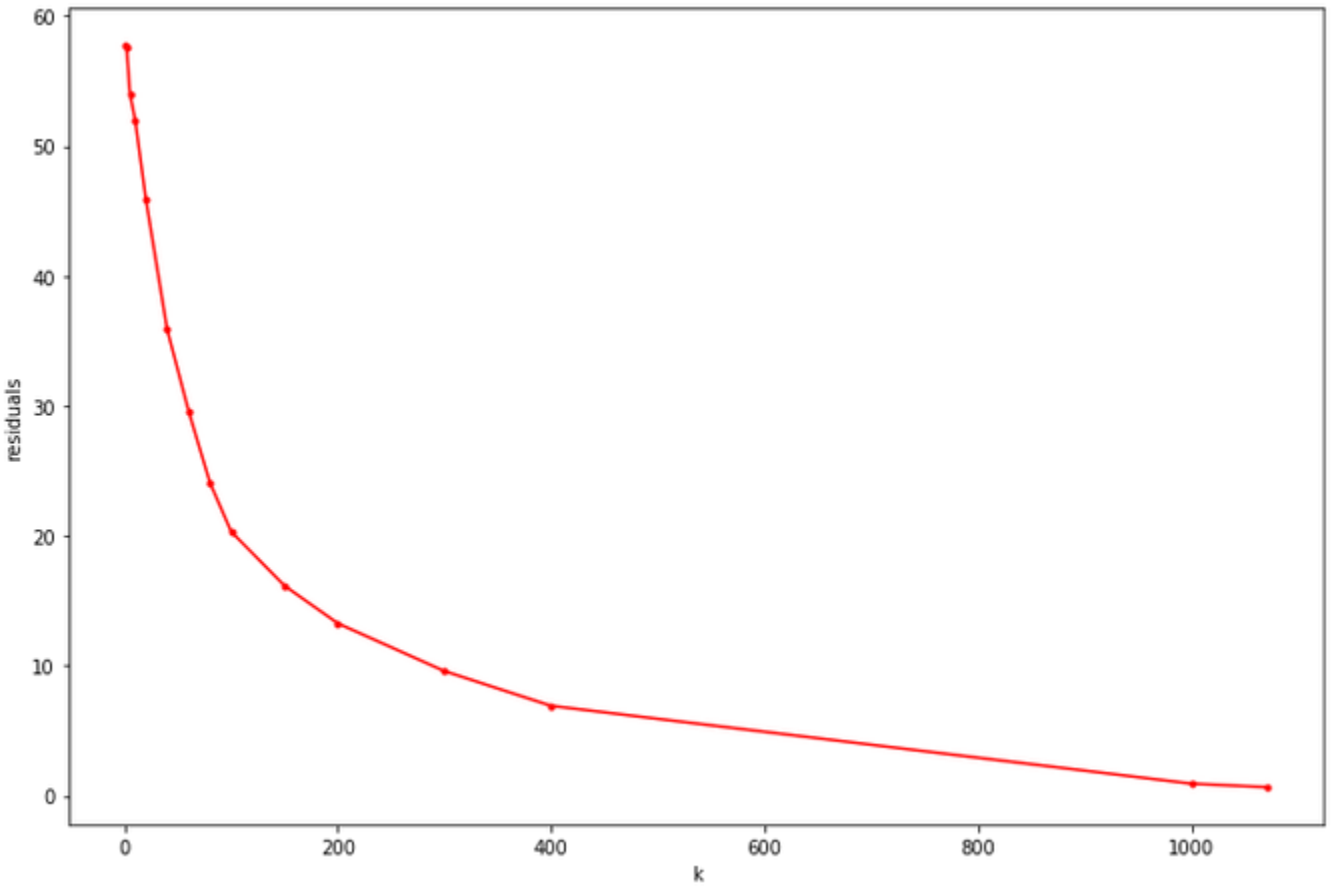


The Reconstruction Error when first k eigenfaces were used

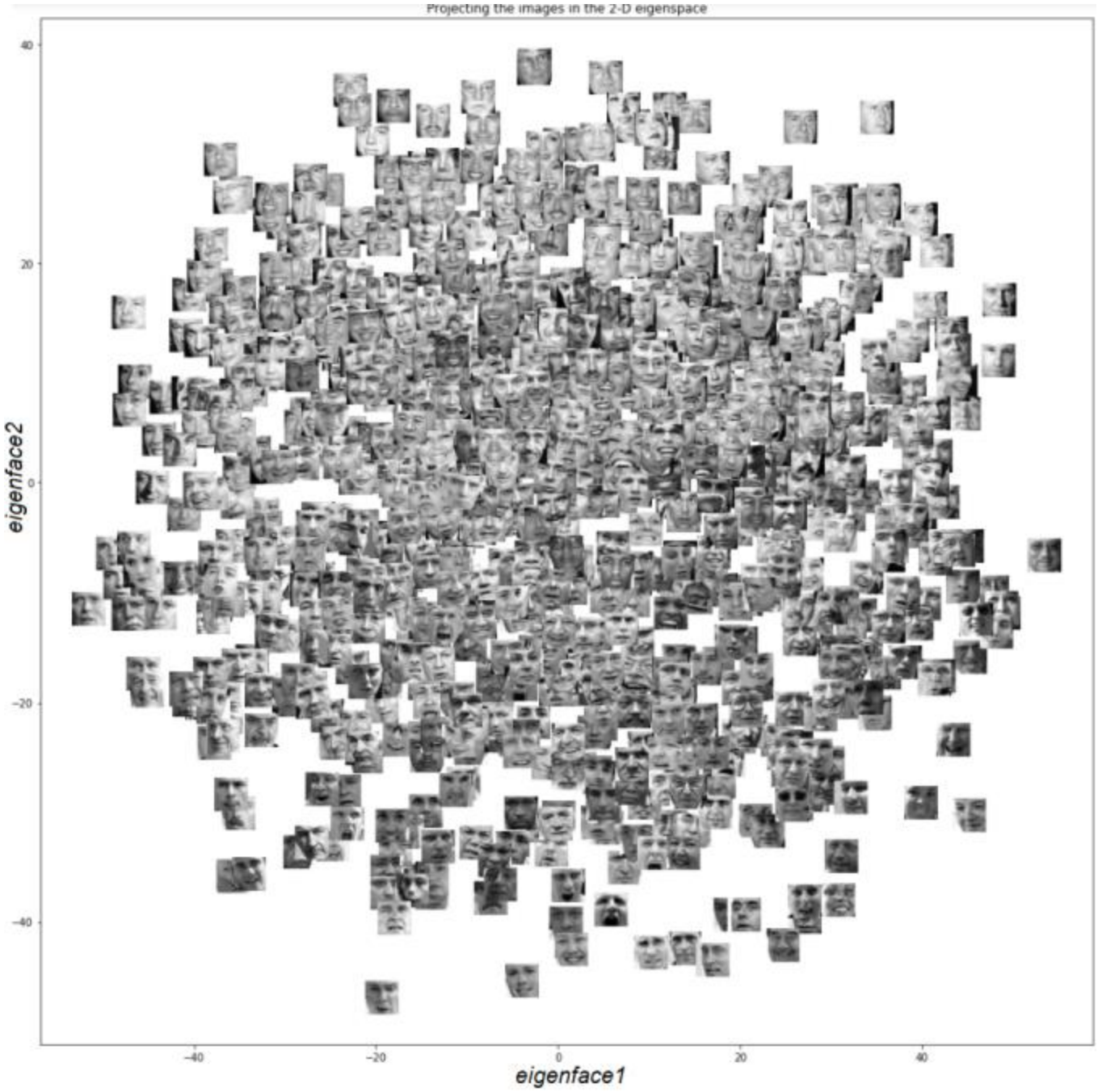


The same is shown for yet another image.


#efaces=1, res=57.804
#efaces=2, res=57.611
#efaces=5, res=54.054
#efaces=10, res=52.01
#efaces=20, res=45.897
#efaces=40, res=35.868
#efaces=60, res=29.624
#efaces=80, res=24.103
#efaces=100, res=20.317
#efaces=150, res=16.154
#efaces=200, res=13.257
#efaces=300, res=9.581
#efaces=400, res=6.908
#efaces=1000, res=0.924
#efaces=1071, res=0.653

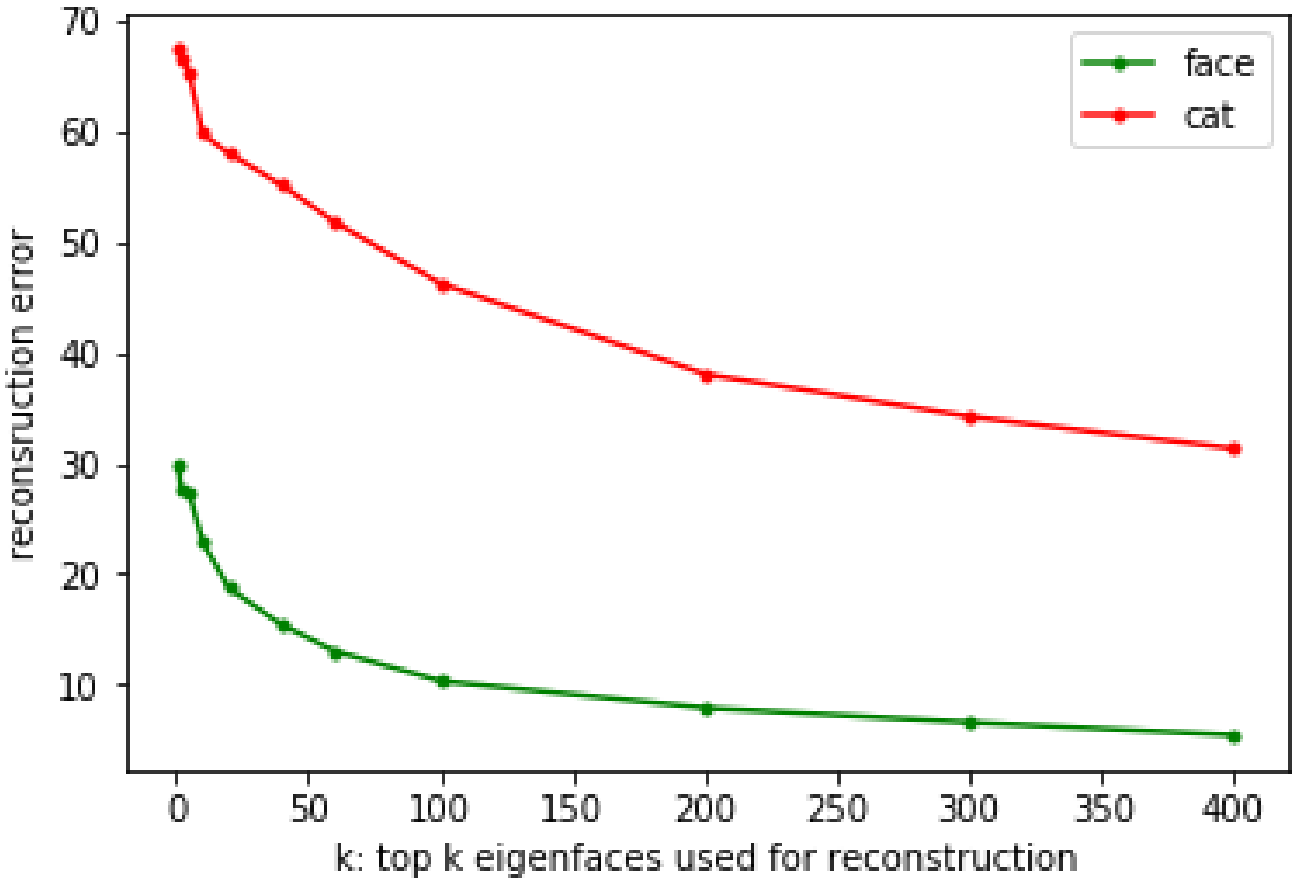


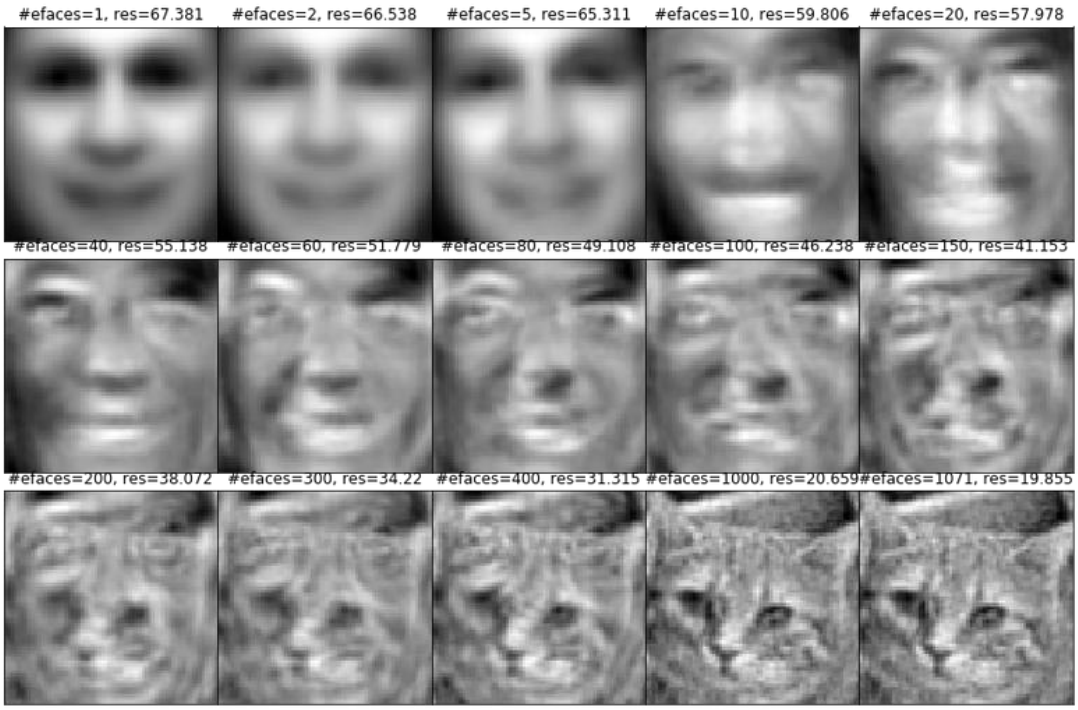
The next figure shows the images projected on and reconstructed from the first two eigenfaces. The similar images stay in close proximity whereas the dissimilar ones (w.r.t. projection on the first 2 dominant eigenvectors) are far apart in the 2D space.



Projection of a Human vs. a Non-Human-Face (e.g. Cat) on the EigenFaces Space

Now let's reconstruct a non-human object (a cat) to the same space spanned by the top-k eigenfaces for different values of k. As we can see from the below figures, the reconstruction error is much higher for the cat, as expected.


face
cat
reconsruction error
k: top k eigenfaces used for reconstruction
70
60
50
40
30
20
10
0
50
100
150
200
250
300
350
400


#efaces=1, res=67.381
#efaces=2, res=66.538
#efaces=5, res=65.311
#efaces=10, res=59.806
#efaces=20, res=57.978
#efaces=40, res=55.138
#efaces=60, res=51.779
#efaces=80, res=49.108
#efaces=100, res=46.238
#efaces=150, res=41.153
#efaces=200, res=38.072
#efaces=300, res=34.22
#efaces=400, res=31.315
#efaces=1000, res=20.659
#efaces=1071, res=19.855

CONCLUSION:

This project successfully helps in lossy compression of an image ie, it helps in compression of an image by reducing the dimensions of images stored, hence saves a lot of storage saved but at the cost of the loss of some data. The precision achieved will not be a 100 percent, hence getting some loss.But this is still considered a very effective method.