MSIS 2631 MACHINE LEARNING

IMAGE COMPRESSION AND RECONSTRUCTION USING PCA

Guided By: Mahmoud Parsian

Submitted By:
Kalyani Rane
Lasya Pathuri

Project Idea:

The main idea of our project is to reduce the dimensionality of a dataset consisting of many dimensions. When we need to tackle data of high dimensions among data with linear relationships, i.e. where having too many dimensions (features) in your data causes noise and difficulties (it can be sound, picture or context). This specifically gets worse when features have different scales (e.g. weight, length, area, speed etc). We do this by reducing the dimensions i.e. the features.

Principal Component Analysis (PCA):

Principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, while retaining the variation present in the dataset, up to the maximum extent possible. The same is done by transforming the variables to a new set of variables, which are known as the principal components and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order.

So, 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.

When we should reduce or change the dimensions:

1. **Better perspective and less complexity:** When we need a more realistic perspective and we have many features on a given data set and specifically when we know that we don't need this much number of features.
2. **Better visualization:** When we cannot get a good visualization due to high number of dimensions we use PCA to reduce it into a shadow of 2D or 3D features
3. **Reduce size:** When we have too much data and we are going to use process-intensive algorithms (like many supervised algorithms) on the data so we need to get rid of redundancy

Some of the useful properties of PCA:

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
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

Step in PCA:

Step 1 - Normalizing 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: We have to reshape the dataset into 2-D, this will result in a 2x2 Covariance matrix.

Step 3 - Calculate the eigenvalues and eigenvectors Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix.

Step 4 - Choosing components and forming a feature vector: We order the eigen values 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.

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 the scaled version of original dataset.

NewData = FeatureVectorT x ScaledDataT

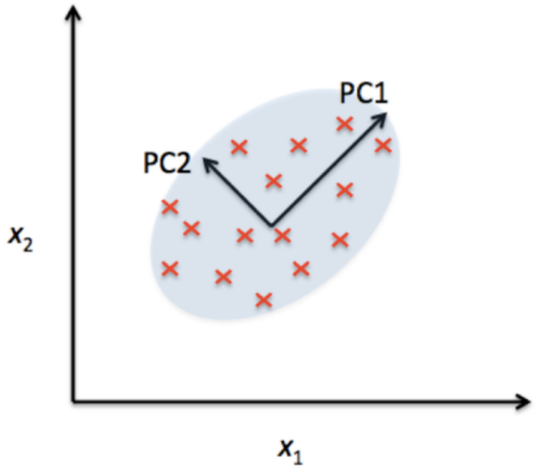
Here,

- 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

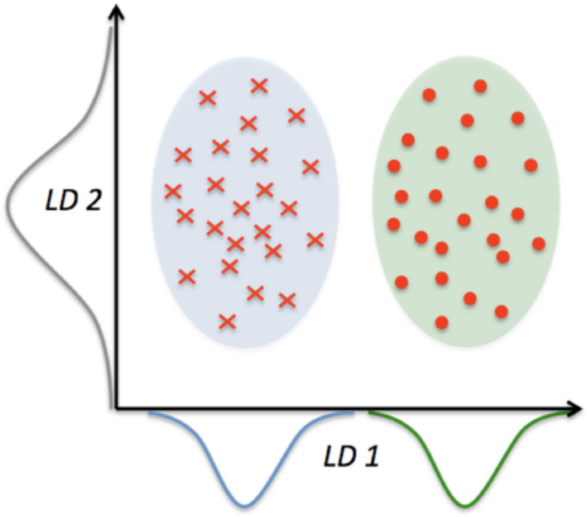
Similar Algorithms as PCA :

Linear Discriminant Analysis (LDA) is also the most commonly dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications.

LDA is **supervised** whereas PCA is **unsupervised** that is, PCA ignores class labels.
We can picture PCA as a technique that finds the directions of maximal variance:

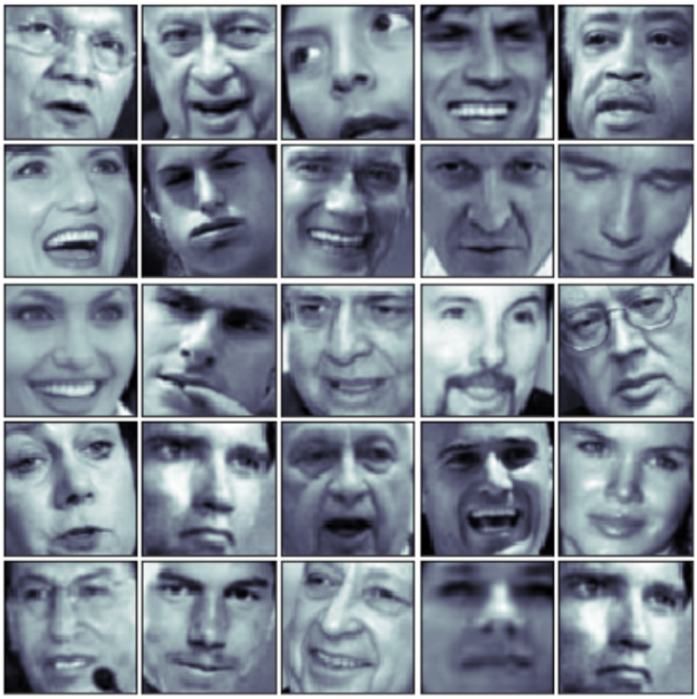


In contrast to PCA, LDA attempts to find a feature subspace that maximizes class separability.



Our DataSet:

Our dataset consists of 1000 images each of size 64*64.



Goal: Compress and reconstruct the image using ‘N’ number of principal components

Below are the libraries used:

- Numpy
- Matplotlib
- Os
- linalg.eig
- sklearn
  - StandardScaler
  - PCA
  - load_digits

Steps Followed:

Initial Data Collection and Reshaping:

1. The first step we have done is **"**cwd = os.getcwd()". OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. os.getcwd()-method tells us the location of the current working directory (CWD)
2. Retrieved images from the directory and stored as an numpy array of shape (1000,64,64)
3. Reshaped the data into 2D (1000,4096) to make it suitable for our analysis

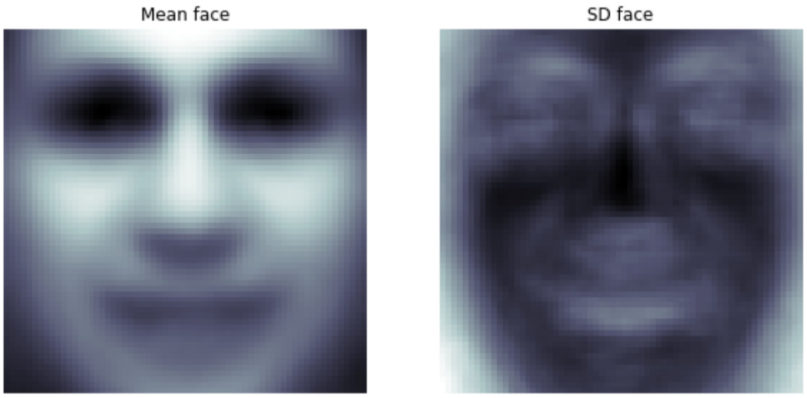
Feature Scaling:

Feature scaling through standardization or normalization is an important preprocessing step for many machine learning algorithms.

4. We have chosen the number of components we need to keep.
   i.e n_comp = 100 (can be varied)

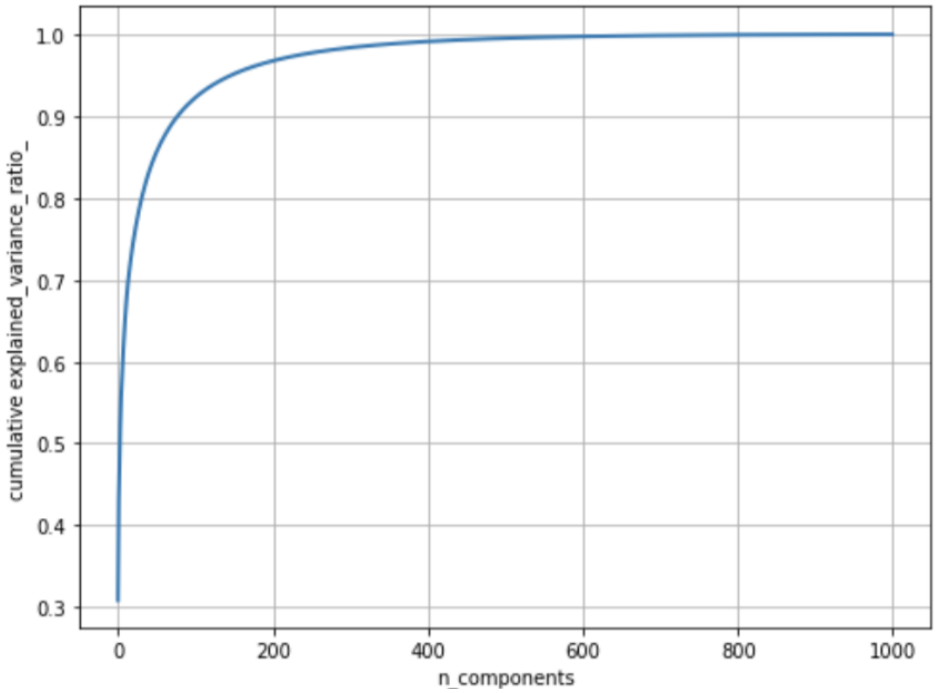
5. **Pipeline:** Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must implement fit and transform methods and the final estimator only needs to implement fit. Here we used StandardScaler, which subtracts the mean from each feature and then scale to unit variance. Now we have created a pipeline object by providing a list of steps. Our steps are — standard scalar and principal components.

6. The interesting part of PCA is that it computes the "mean" and "Standard Deviation" face, which can be interesting to examine. We have computed both the values and displayed the output face.



Number of Principal Components Vs Explained_Variance_Ratio:

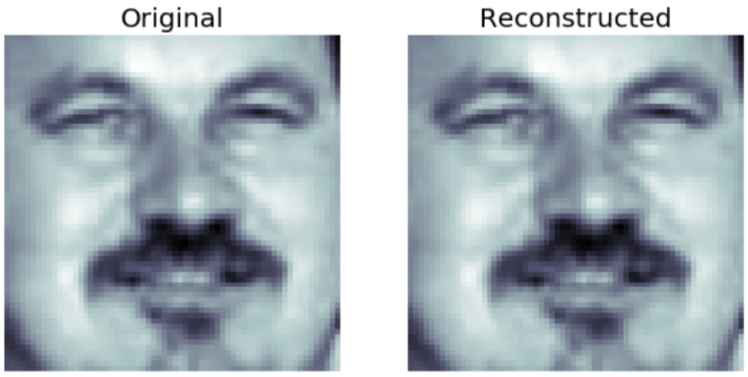
7. We have Plotted a graph to clearly visualize the amount of variance in N number of Principal Components



Reconstructing the Images:

8. We have used **inverse_transform ()** to get the images back. It transforms data back to its original space. It obtains the projection onto components in signal space you are interested in.

9. We can change the n_compo and repeat the process again.



Computing PCA Manually (Without Sci-kit):

Data Manipulation:

1. Converted the image matrix to 4096.

2. Calculated the mean of the images matrix as
   imagesMean =(faces.mean(axis=1)).reshape(-1,1)

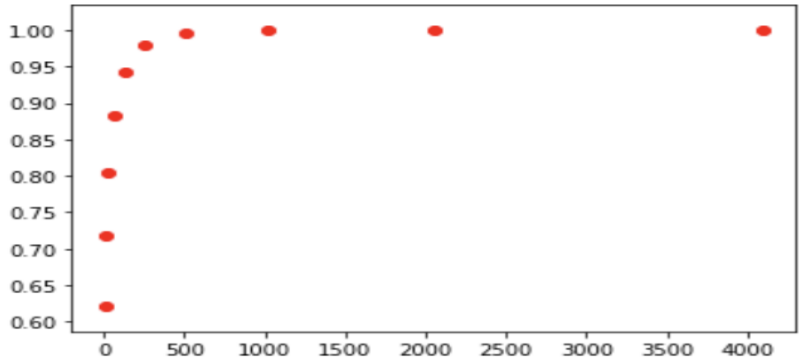
3. Instead of using pipeline, we have normalized the data as data- mean data.
   normalisedImages = faces - imagesMean

4. Calculated covariance matrix as a product of normalized data and its transpose.
   covarianceMatrix = (normalisedImages).dot(normalisedImages.T)

5. Computed eigenvalues and eigenvectors using linalg.eig. It computes the eigenvalues and right eigenvectors of a square array.
   eigenValues, eigenVectors = np.linalg.eig(covarianceMatrix)

6. Sorted the eigenvalues in descending order.

7. Graph of number of principal components vs explained_variance_ratio for manually computed eigenvalues.

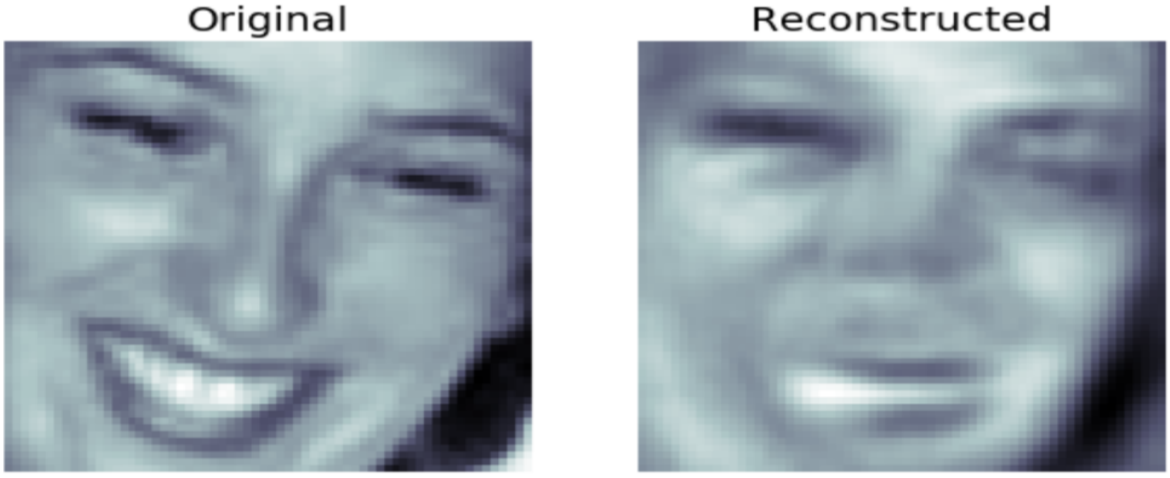



```
For k = 8        -------->        62% of variance is covered.
For k = 16       -------->        71% of variance is covered.
For k = 32       -------->        80% of variance is covered.
For k = 64       -------->        88% of variance is covered.
For k = 128      -------->        94% of variance is covered.
For k = 256      -------->        97% of variance is covered.
For k = 512      -------->        99% of variance is covered.
For k = 1024     -------->        100% of variance is covered.
For k = 2048     -------->        100% of variance is covered.
For k = 4096     -------->        100% of variance is covered.
```

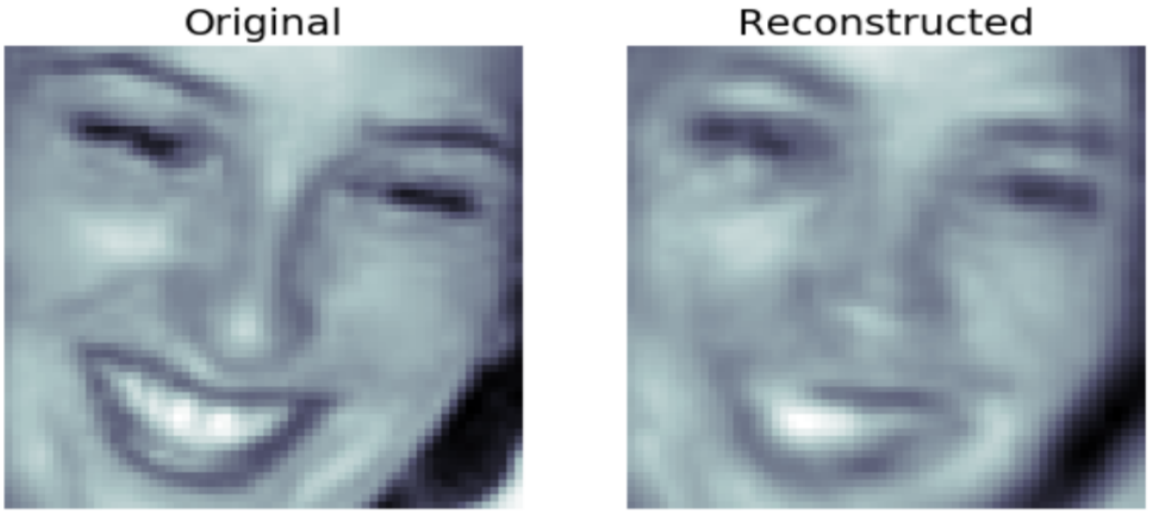
Now, we can use these values obtained to reconstruct the image back as earlier

Results:

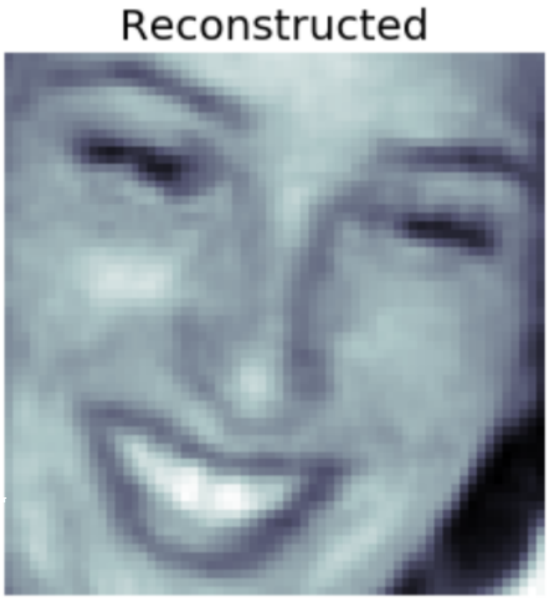
For n_comp = 50



For n_comp = 100

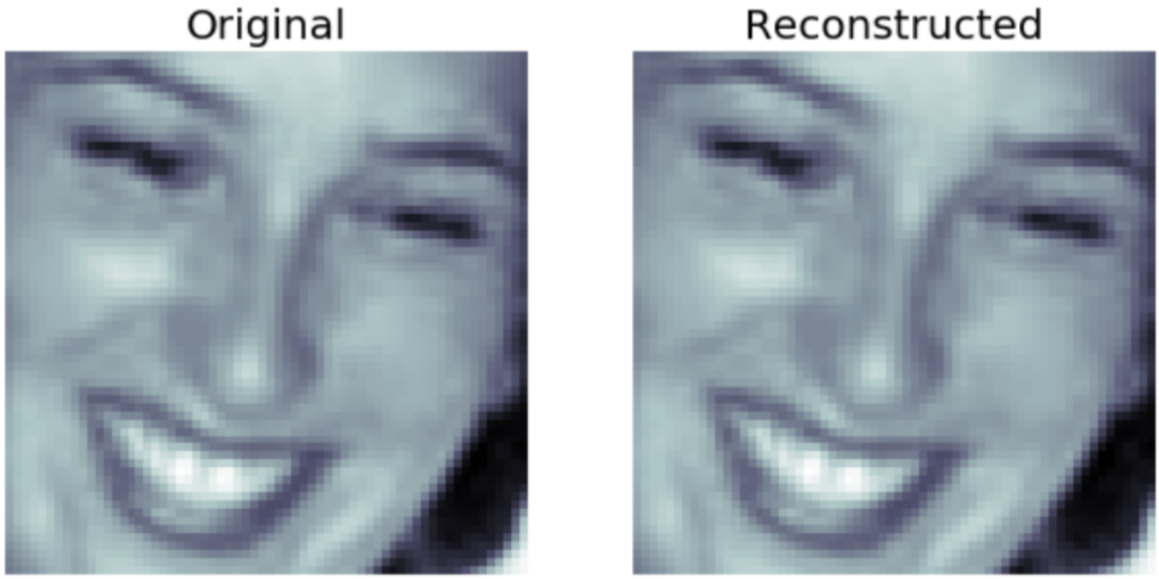


For n_comp = 500



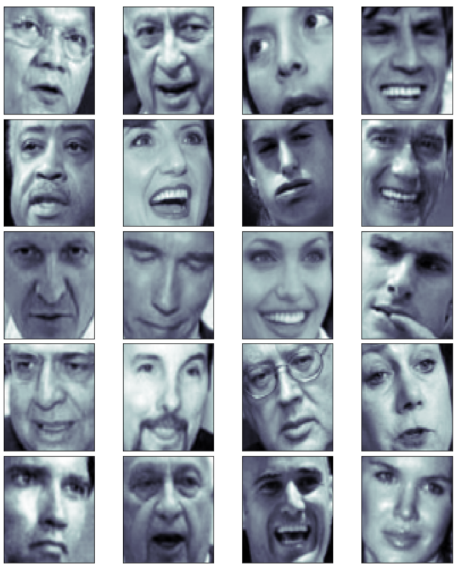
For n_comp = 700

For n_comp = 1000



For n_comp = 100

For n_comp = 1000



We can clearly see that as n_comp increases, image reconstruction quality is improved.

Conclusion:

PCA is one the popular dimensionality reduction techniques used by professionals in industry. It helps to build efficient models by reducing the feature set, boosting learning rate and diminishing computation costs by removing redundant features. By applying PCA to a set of images, we successfully reduced the dataset size and also reconstructed these images from the compressed versions of them. The reconstructed images were quite close to the original ones and the quality increased as the number of PCs increased.

References:

https://www.quora.com/When-and-where-do-we-use-PCA
https://www.dezyre.com/data-science-in-python-tutorial/principal-component-analysis-tutorial
https://sebastianraschka.com/Articles/2014_python_lda.html
https://sebastianraschka.com/faq/docs/lda-vs-pca.html
https://towardsdatascience.com/a-simple-example-of-pipeline-in-machine-learning-with-scikit-learn-e726ffbb6976
https://stackoverflow.com/questions/36566844/pca-projection-and-reconstruction-in-scikit-learn