**Dimensionality Reduction for different data formats: Implementation of various techniques using Python**

# Abstract

There are many formats in which data can be stored. The various data formats being Images, text, numerical. All these forms of data have numerous dimensions; hence we require dimensionality reduction.

To operate on any format of data, dimensionality reduction makes the job easy. There are several techniques for dimensionality reduction. Dimension is the feature of the given data.

For example, for a flower feature is petal length, which is also its dimension in terms of data.

Dimensionality reduction can be done by feature selection. Feature selection approaches try to find a subset of the input variables.

Data analysis such as regression and classification can be done in the reduced space more accurately than in the original space for higher dimensional data.

To avoid the curse of dimensionality, we reduce the dimensions of the data. Feature extraction and dimension reduction can be combined into a single step using various techniques.

In mathematical terms, Dimensionality reduction can be described as process of deriving a set of degrees of freedom which can be used to reproduce most of the variability of the dataset. Dimension reduction techniques which are broadly used:

* 1. PCA-Principle Component Analysis

PCA: It is the most widely used technique. It provides a sequence of best linear approximations to a given high-dimensional observation. The subspace model led by PCA captures the maximum variability in the data. However its effectiveness is limited by it’s global linearity. In-fact, in all the classical dimensionality reduction techniques characterizes only linear subspaces and hence suffers from this drawback.

* 1. LDA-Linear Discriminant Analysis

LDA: It is a popular technique for reduction of dimension. It considers the important features which can be used to distinguish the entire datapoints in the dataset. Built in function is present in Sklearn library under discriminant\_analysis module.

* 1. FFT-Fast Fourier Transformations

FFT: These can be applied using built in functions in python. It is present in numpy library under the fft module.

* 1. Eigen value of Covariance matrix

It is a manual method for reducing the dimensions up to the required feature for further applications. The calculations that are performed in this method can be done with the help of numpy library.

In this report, all the mentioned techniques are compared using four different datasets. The results obtained from the techniques for the datasets are pretty satisfactory. It also shows which techniques are best for which type of data format. The further application used for the technique in this report are matching/ finding the most similar data point from the dataset, hence identifying the data point as well. Using the Sklearn metrics, results are analyzed.

After the analysis of the techniques, we will be looking at the different types of   
transformations/representation of data. Since, we know there are many forms of representing data, it is important for us to understand which one to use when and where in order to get optimum results for the operation that is to be performed or any algorithm that it is required for. We will also see how data is represented and what are changes involved for that transformation to take place.

# Principal Component Analysis for Dimensionality Reduction

Given a set of data points of n dimensions, PCA finds the linear sub space of dimensions lower than n such that the data points lie mainly on this linear subspace. The derivation of PCA is defined as: Let the t centred observations Xi i=1,2,.,t be stacked columns of an nxt matrix X, where n is the dimensionality of the observations. Choose principal component U1as a linear combination of X so that it captures maximum variance. That is, choose

U1=WTX

So that variance(U1)=variance(WTX)is maximum, where WT =[W1,W2,W3,. ,Wn].

Choose larger values for the components of W, it follows the determination of the optimum weight vector required normalization of its magnitude. The optimization problem:

WTSW should be maximized Such that, WTW=1

Where S is then x n sample covariance matrix of X. Introducing LaGrange’s multiplier, the problem becomes:

L(W,a)=WTSW–a(WTW-1)

Differentiating with respect to W and equating it to zero, we have:

WTSW = WTaW = aWTW = a

&

SW=aW

This implies WTSW is maximum if and only if W is an Eigen vector corresponding to the largest Eigen value. This shows that the first principal component is given by the normalized given vector corresponding to the largest Eigenvalue of the covariance matrix. Similar argument can show that the d dominant Eigenvectors of the covariance matrix X determine the first d principal components.

In fact the d principal components can be determined from the first d columns of the left singular matrix X, i.e. from the first d columns of U of X = UqVT.

Other than PCA, other techniques are also present in the literature. There are several other variations of PCA, they are Kernel PCA, Dual PCA, Metric Multi dimensional Scaling, Semi Definite Embedding etc. Kernel PCA is designed in order to tackle the problem of PCA dealing with non linear data formats.

# Linear Discriminant Analysis for Reduction

LDA is a pre-processing step used in Machine Learning and applications of pattern classification. Here, first we calculate the separability between classes which is the distance between the mean of different classes. This is called the between class variance.

Mean=Sum(x)/Nk

∑2=Sum((x-Mean)2)/(N-k) Mean = mean value of x for a class

∑2 = Variance across all inputs x N = number of instance

k=number of classes M=mean of input x

Next, we find out the variance of data points within the class; this is called within class variance. Then, we construct the lower dimensional space which maximizes the between class variance and minimizes the within class variance.

LDA models make predictions based on the use of Baye’s theorem which estimates the probability of the output class given the input.

P(Y=x\X=x)=[(Plk\*fk(x))]/[sum(Pll\*fl(x))]

x=input

k=output class

Plk=Nk/n or base probability of each class observed in the training data fk(x)=estimated probability of x belonging to class k

Using the probability values for any given input the class is determined, based on the probability value being highest for the assigned class.

LDA works when the measurements made on independent variables for each observation are continuous quantities. When dealing with categorical independent variables, the equivalent technique is Discriminant correspondence analysis.

Discriminant analysis is used when groups are known prior to building the model. Each case have a score on one or more quantitative predict or measures, and as core on a group measure. In simple terms, Discriminant function analysis is classification – the act of distributing things into groups, classes or categories of the same type.

**Calculations involved in LDA:**

(For clear understanding)

There are two variances which are considered in LDA process:

* Between class variance
* Within class scatter

Between class variance is defined as the total variance between all the classes present in the dataset. Number of between class variance = Total number of classes in the dataset.

Within class variance is defined as the total variance between the data points pertaining to the same class. The total number of within class variance = Total number of classes in the dataset.

Based on these two the axis is generated to maximize the between class variance and minimize the within class variance.

Between class variance = Sb = (m1-m2)(m1-m2)T

Within class variance = Sw = ∑ ( ∑(xi - m1) (xi - m1)T )

The outer summation indicates that the sum has to be repeated for the classes.  
For example, if there are 5 classes, then the multiplication of 2 matrixes has to  
to be done for all 5 classes.

In LDA, we find a discriminatory vector using which the data can be visualized in a smaller dimensional

space.

This is the reason we can only choose (n-1) value for components. [n = number of classes in dataset].

So, how to determine how many discriminatory vectors can be used for representing the data?

It can simply be answered by how many non zero discriminatory vectors we obtain for the corresponding   
Eigen values.

The eigenvalue problem of LDA can be defined as:

Sb V1 = (Eigen value) Sw V1

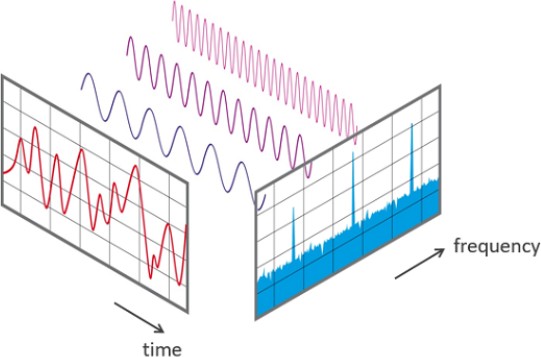
Sb = Between class variance

Sw = Within class variance

V1 = Vector

Eigen value = To be found

# Fast Fourier Transformation for Reduction

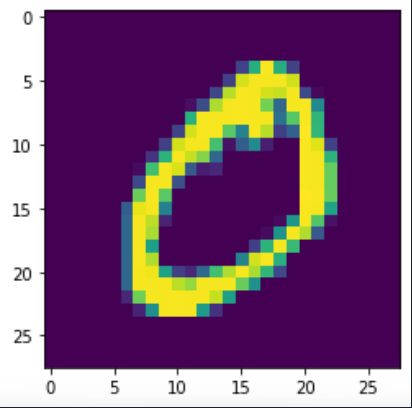
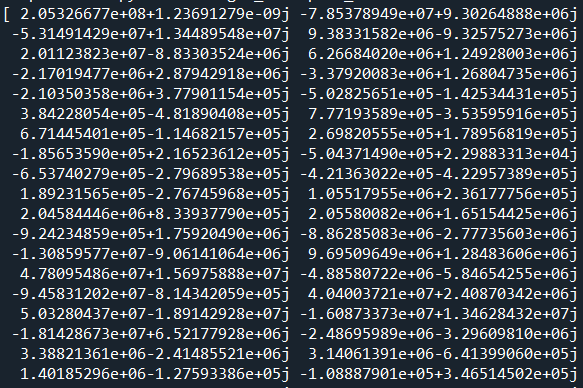


FFT is a important measurement technique in science of audio and acoustic measurement. It converts signal into individual spectral components and thereby provides frequency information about the signal. FFT is an optimized algorithm for the implementation of Discrete Fourier Transformation(DFT).

As observed from the diagram over the time period measured, a signal contains 3 distinct dominant frequencies.

Though FFT is suitable for periodic signals only, we can also use it for other data forms as well. This is for signals.

In case of data present in numerical, text, image format (i.e. numerical data itself),we can convert the data into complex numbers after undergoing FFT. It is a built in function from numpy library. The transformation example:



After applying FFT transformation, we obtain the text on the right.

(Note: the data shown is not the entire transformed data, but, just a small portion for representation purpose)

The data is transformed only after obtaining the key components and then it is used for comparing images. The data is transformed using fft2 () function in python. The transformed data can be transformed back if needed using ifft2 () function in python present in the Numpy library.

The data obtained even after inverse transformation is complex, which can be dealt with using various mathematical operations to obtain the original image back from the transformed data. Using the transformed data’s absolute form, we can compare with the new input image or other data form and apply the algorithm as required. The example considered is from MNIST dataset after constructing the images from the dataset.

# Eigen Value of Covariance matrix method for Reduction

In this method, we use the key components which can be used to re-construct the entire dataset. These components are features from the dataset. In case of MNIST dataset, (which is pre-processed dataset as we require such data) we would consider the main pixels which can be used for re-constructing the entire dataset back. The steps involved in this technique are:

1. Obtain the data points from the dataset. (SayimagesI1, I2, I3 ,., In).
2. Represent every image Ii (NxN)as a vector Vi(N2x1)
3. Compute the average image vector
4. Subtract the mean image vector from the entire dataset. Mean matrix obtained is of the size N2 x M
5. Compute the covariance matrix C

C=1/M(meann x meannT)(N2 x N2 matrix)

1. Compute the eigen vectors for the covariance matrix.

Compute the covariance matrix in such a way that the computation is simpler

i.e. use meannTxmeann (MxMmatrix)

1. Consider the top K eigen vectors for the largest Eigen values obtained.
2. Using these K vectors we can reconstruct all the images from the dataset or these can be used for further applications.

After reconstructing the images, we can use them for image comparison in a faster and more optimized way. Even if we don’t reconstruct the images, the key features can be compared to determine the class label for the image.

For other data formats, we apply the same technique, that we consider the key components that will be useful for reconstructing the data back.

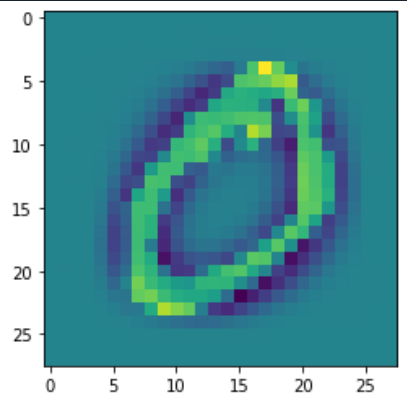
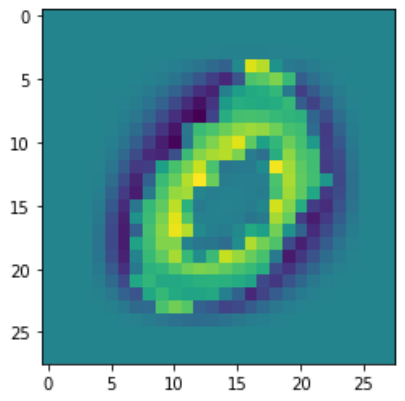
We can calculate the distance between the new input and the different class Eigen vectors for determining the data point belongs to which class. The minimum distance will be considered for assigning the class to the input data. This is a optimized way of classifying data.

There are many other applications after forming the eigenvectors of data points. We can use it for reconstructing data points with a few features available. We can also differentiate new data points into class according to the present class labels.

All the images are represented using compressed form of image.

For example, in the zero digit representation using compressed images, we obtain:

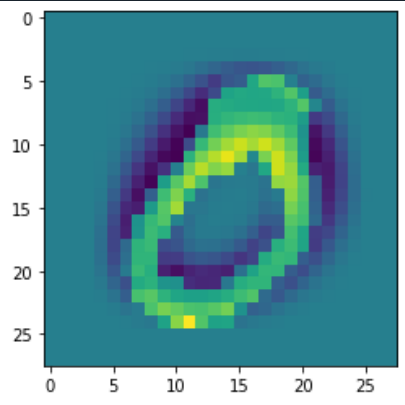
The image is compressed and can be reformed using the below images:

A picture containing screenshot, colorfulness, diagram, plot

Description automatically generated

+K2\*

K1\*

=

+K3\*

+ ……..

Not only the sample image, but also the remaining images of set containing digit 0 can be represented using the image.

Note: These aren’t all the images used for representing the image. This is just a sample.

Note: K1, K2, K3 are the corresponding Eigen values to the corresponding Eigen vectors.

The images used for reconstructing the dataset are also known as Eigen faces of the dataset.

# Datasets Used

We have considered three datasets mainly for the analysis.

MNIST dataset, that is available in Kaggle. This is a smaller version of the NIST dataset. NIST dataset contains 1,20,000 images for training and 20,000 images for testing. MNIST contains 60,000 images for training and 10,000 images for testing the model. The format of data is images. There are a total of 10 classes available.

IRIS dataset is considered which has 4 features pertaining to flowers- Sepal\_Length, Sepal\_Width, Petal\_Length and Petal\_Width. There are 150 samples available in this dataset. The format of data is numerical. There are total of 3 class labels.

GENE-LEUKEMIA dataset is also considered which has 22,384 features with a few sample space of 64. The format of the data present is in numerical format. There are total of 6 classes present in this dataset which are texts.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No. | Dataset | #Samples | #Features | #Classes |
| 1 | MNIST | 60,000 | 28x28=784 | 10 |
| 2 | IRIS | 150 | 4 | 3 |
| 3 | GENE | 64 | 22,384 | 6 |

We will be applying PCA – Principle Component Analysis, LDA – Linear Discriminant Analysis , FFT – Fast Fourier Transformation , Eigen Value of Covariance method for all the three datasets and analyzing the

* 1. Accuracy of each method – For this we will be considering the MNIST dataset as there are many images, which will depict the difference in the accuracies of the methods more precisely.
  2. Optimum dataset size for each technique – For this analysis, we will be considering the IRIS dataset, as the dataset has many samples and few features, which will make the analysis of dataset size more precise.
  3. Time analysis of each technique – For this we will be considering the GENE-LEUKEMIA dataset as there are 22285 features available, which will help us to understand the time taken by each technique to select the key features according to the data provided.

# : In detail about the used Datasets

1. MNIST dataset

This dataset has 4 datasets under it. MNIST dataset is a smaller version of NIST dataset. It contains 60,000 images under the first the 1st dataset. The 2nd data set contains the labels that can be used to train the model. The 3rd dataset contains the testing images which are a total of 10,000 and the 4th dataset contains the labels of the testing images which can be used to determine the accuracy of the model build.

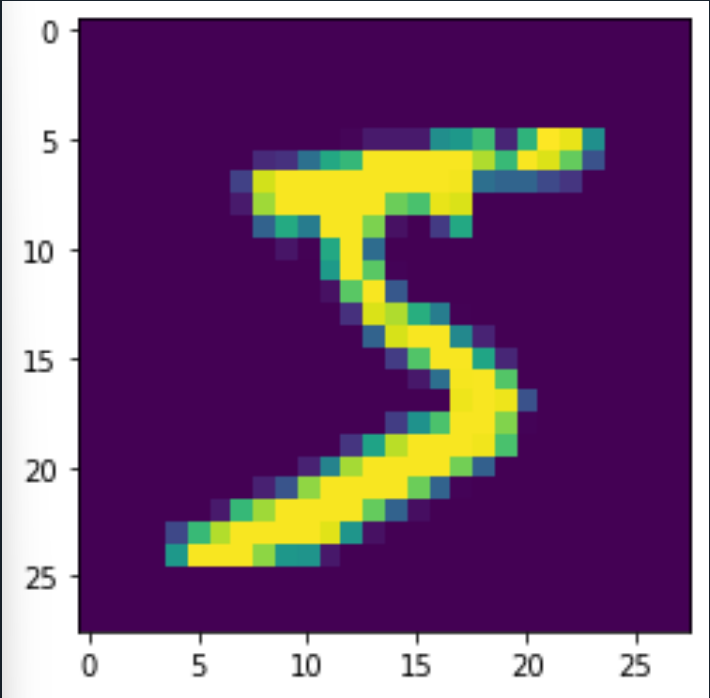
The dataset is a collection of handwritten digit images which are already pre-processed

i.e., all the images are center aligned. The size of each image is 28x28 (Digits range from 0 to 9). Hence, there are 10 classes present in the mentioned dataset.

All the 4 datasets are in Binary form and hence we need to first obtain the images from the dataset and then use them for building the model. The data format is in images.

This dataset is very good for building and learning ML algorithms.

This dataset was developed by Yann LeCun, which is mini version of NIST dataset developed by Chris Burges and Corinna Cortes. The mini version was developed using bounding- box normalization and centering.

Sample image reshaped into 28x28:

1. IRIS flower dataset

This is a small dataset used in analyzing various Machine Learning Algorithms built by the programmer. This dataset contains 150 samples. These samples have 4 features each. There are only 3 classes in it namely, Iris-setosa, Iris-virginica, Iris-versicolor. The4featuresaresepal\_width, Sepal\_Length, Petal\_Width, Petal\_Length. Using these 4 features we can determine the class of the flower. But still, we reduce the dimensions of the dataset to determine the class label of any given data point.

The dataset used here is in CSV format obtained from Kaggle. It is a preprocessed dataset. The classes are evenly divided, meaning each class contains 50 samples in it. The data format is in numerical.

1. GENE-LEUKEMIA dataset

This dataset contains 22285 features and only 64 samples in it. It contains 5 class labels named AML, Bone Marrow, Bone\_Marrow\_CD34, PB, PBSC\_CD34. As we can see the features are too many and can lead to delayed output and can lead to over fitting of data. The data format is in numerical format.

The dataset is in CSV format obtained from Kaggle. It is a pre processed dataset. The features are present in numerical and text format which can be used for classification of data using various techniques like KNN, LDA, etc. But to apply KNN, we need to reduce the dimensions of data to lower dimensions which can be visualized.

# Related Work/History

Morse code, invented in 1838 for use in telegraphy, is an early example of data compression. Modern work on data compression began in the late 1940s with the development of information theory.

An optimal method for compression was then found by David Huffman in 1951. In themid1970s, the idea emerged of dynamically updating codewords for Huffman encoding, based on the actual data encountered. And in late 1970s, with online storage of text files becoming common, software compression programs began to develop, almost based on Huffman coding.

Then came the issue of lossy compression, which is being encountered and dealt with even today.

The lossy compression is achieved at a ratio of 3:1 i.e., original data being 3 times the compressed data. But the lossless compression is still approximately in the ratio of 2:1

i.e., the compressed data is ½ of the original data.

There are numerous techniques now developed for enhanced data compression. The main idea is to reduce the dimensions of data in order to compress it. As the data needs to be reconstructed back from the compressed data without any loss, compression needs to be done without a loss.

In the present era, zipping and unzipping of files is a common operation. But more optimization is possible for compression, on which still research is going on.

1. **Algorithms**

#Algorithm for analyzing the accuracy score of different techniques Vs number of components considered for reconstructing the images and hence finding the most similar image in MNIST dataset

Import the required libraries.

Import the 4 datasets from MNIST.

Divide the dataset based on handwritten digits using the label dataset.

Create X\_train, Y\_train, X\_test, Y\_test

Algorithm Eigen\_value\_of\_Cov ():

1.Find the mean of all the datasets.

2.From the entire dataset, subtract the mean image and store it.

3.Using this create Covariance matrix and find eigenvalues and eigenvectors.

4.Consider top n vectors, these can be named as the number of components.

Repeat steps 1-4 for all the formed datasets.

5.Now, project the compressed data using dot product of datacenter and eigenvectors

6.Consider a new input image, now create a new center by subtracting input image and mean image.

7.Create new projection of data using the input image and eigenvector of all the formed datasets.

8.Calculate the distance between the projection and new\_projection using cdist built in function (in python).

Repeat steps 5-8 for all the formed datasets.

Using all the distances obtained,

calculate the minimum distance between all the new projections and projected data.

Place the value of the digit obtained from the dataset as the index,

find the most similar image from the training dataset.

Hence, we can calculate the remaining parameters like accuracy score etc.

Repeat all the steps in the Algorithm with different number of components and analyze the accuracy score Vs components

Algorithm LDA():

import Linear Discriminant Analysis from Sklearn. Discriminant\_analysis module.

Form the LDA model to compress the images based on number of components.

Using the model fit the training dataset and label dataset.

Using LDA.predict(), predict the X\_test label, find the most similar image and

hence it can be compared with Y\_test label dataset.

Repeat all the steps in the Algorithm with different number of components and analyse the accuracy score Vs components

Algorithm PCA ():

import PCA from Sklearn. decomposition module.

Select the number of components.

Using that, transform/ compress the data from each formed dataset.

1.Find the mean of compressed data form.

2.Using input image, calculate the new\_projection of dataset by

subtracting the mean image from input image.

3.Calculate the minimum distance from the new projection and old projection.

Repeat steps 1-3 for all datasets.

4.Using the minimum distance find the most similar image from the training dataset.

Repeat all the steps in the Algorithm with different number of components and analyse the accuracy score Vs components

Algorithm FFT ():

import fft2 from the numpy.fftmodule

import ifft2 from the numpy.fftmodule

1.Transform the compressed data obtained from Eigen value Of Covariance method.

Repeat the above step for all datasets.

2.Find the new center by subtracting the mean image from each input image and project the data

3.Name this projection as new projection

4.Calculate the distance between the previous projection and new projection for all datasets.

5.Compute the minimum distance from all the datasets and consider the index from the training\_set.

6.The other parameters can be calculated based on this process as well.

Repeat all the steps in the Algorithm with different number of components and analyze the accuracy score Vs components

For all the techniques:

Calculate the accuracy score using accuracy\_score from sklearn.metrics

Calculate the data loss obtained at each technique using the probability values of the predicted class.

Data loss can be calculated using entropy gain method, i.e.   
[-log (probability values) \* (Y\_test values) ]

#Algorithm to find the optimum dataset size using IRIS dataset

import the essential libraries.

Load the CSV file using pandas.read\_csv()

Consider any value to begin with for training set size.

Construct X\_train, X\_test, Y\_train and Y\_test datasets from the IRIS.csv dataset.

Using groupby () function group the datasets with same class label.

Algorithm LDA():

import LinearDiscriminantAnalysis from Sklearn. Discriminant\_analysis module

Build a model and fit the X\_train dataset.

Using LDA.predict() predict the Y\_label values for the specified number of components.

Repeat all the steps in the Algorithm with different training set sizes and analyze the accuracy score Vs components.

Algorithm PCA ():

import PCA from sklearn.decomposition module.

Using the 3 datasets obtained from using groupby function, transform the dataset for specified number of components.

For the new input data, project the data by subtracting the mean of that dataset

Find the distance between the input datapoint and all classes.

Predict the class labels using the minimum distance, which can be calculated using cdist()

#cdist is a built in function in scipy.spatial.distance module.

Repeat all the steps in the Algorithm with different training set sizes and analyse the accuracy score Vs components.

Algorithm FFT():

import fft2, ifft2 from numpy.fft module.

Consider the compressed data and apply the fft2 transformation on all datasets.

Inverse transform the transformed dataset using ifft2 function.

Using the new input image calculate the distance between the new projection and

Using the minimum distance from each class, obtain the class labels.

Repeat all the steps in the Algorithm with different training set sizes and analyse the accuracy score Vs components.

Algorithm Eigen\_value\_of\_Cov ():

Obtain the mean of all the 3 datasets obtained from groupby function.

1.Calculate the covariance matrix for the dataset.

2.Compute the eigenvalues and eigenvectors of the covariance matrix obtained.

3.Consider the top n eigenvectors.

4.project the data based on the number of components that you want to consider.

Repeat steps 1-4 for the all the 3 datasets.

5.Subtract the mean image from each input image.

New projection is projection of input data-mean image.

6.Calculate the distance between the new projection and old projection.

7.Consider the minimum distance from the classes and obtain the label of the input datapoint.

Repeat steps 5-7 for all the input datapoints from the testing dataset.

Repeat all the steps in the Algorithm with different training set sizes and analyse the accuracy score Vs components.

For all the techniques:

Calculate the accuracy score using accuracy\_score from sklearn.metrics

#Algorithm to analyze the time taken by each technique using GENE-LEUKEMIA dataset

import the essential libraries.

Read the csv file of Gene-leukemia and obtain training and testing set

Split the datasets into X\_train, Y\_train, X\_test, Y\_test

Using groupby function divide the dataset based on the class labels.

Algorithm LDA ():

import LinearDiscriminantAnalysis from Sklearn. Discriminant\_analysis

Start time using time () from time library.

Select the number of components.

Using the selected number of components, build the model

Fit the model using X\_train and Y\_train.

end time using time () from time library.

Using LDA.predict(), predict the class labels.

Repeat all the steps in the Algorithm with different number of components and analyze the time taken Vs components.

Algorithm PCA ():

import PCA from Sklearn. decomposition

Select the number of components.

start time using time () from time library.

Build the model for PCA.

Transform the data from all the 6 datasets.

For finding the most similar datapoint, we need to project this data

and find the new projection of data using (input data-mean datapoint)

Find the minimum distance between the projection of each class and datapoint projection.

Using the minimum distance obtained, obtain the class labels.

end time using time () from time library.

Repeat all steps in the Algorithm with different number of components and analyze the time taken Vs components.

Algorithm FFT ():

import fft2, ifft2 from NumPy. Fft module.

Start time using time () from time library.

Transform the compressed data and project it after inverse transforming it.

Name it as projection.

Using the input datapoint, find the new data center by subtracting the mean data from input datapoint.

Project the data as new\_projection.

Using the distance between the projection and new\_projection find the minimum distance between class and input datapoint.

Predict the class labels using the minimum distance.

end time using time () from time library.

Repeat all steps in the Algorithm with different number of components and analyse the time taken Vs components.

Algorithm Eigen\_value\_of\_Cov ():

Obtain the mean of all the 6 datasets obtained from groupby function.

Start time using time () from time library.

1.Calculate the covariance matrix for the dataset.

2.Compute the eigenvalues and eigenvectors of the covariance matrix obtained.

3.Consider the top n eigenvectors.

4.project the data based on the number of components that you want to consider.

Repeat steps 1-4 for the all 6 datasets.

5.Subtract the mean datapoint from each input datapoint.

New projection is projection of input data-mean image.

6.Calculate the distance between the new projection and old projection.

7.Consider the minimum distance from the classes and obtain the label of the input datapoint.

Repeat steps 5-7 for all the input datapoints from the testing dataset.

end time using time () from time library.

Repeat all steps in the Algorithm with different number of components and analyze the time taken Vs components.

# Observations

* 1. **5.1 : Analyzing the number of components required for compression.**

Considering the components that are required for reconstructing the entire dataset, there can be variation according to the data present in the dataset. As we can reconstruct the entire dataset using few or more components resulting in different accuracies. Consider the MNIST dataset with 784 features (of 60,000 images) we can consider various numbers of components to reconstruct all the images.

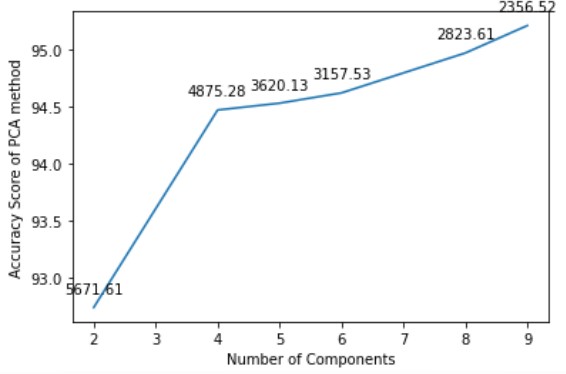
We can find the most similar image using the compressed images. This is performed in order to reduce the time required for the comparison between images. As we know comparing the image pixel by pixel is going to take greater time, hence the consider only the required components.

The numbers in the graph depict the loss of data while compressing the images and using them for the given task. The numbers in the graph are original loss/100.

# Using Eigen value of Covariance method:

As we can observe the accuracy of obtaining similar images from the dataset increases as we consider more components for reconstructing the dataset. The accuracy obtained for 300 components is 96.56%.

# Using PCA built-in function from Sklearn library:



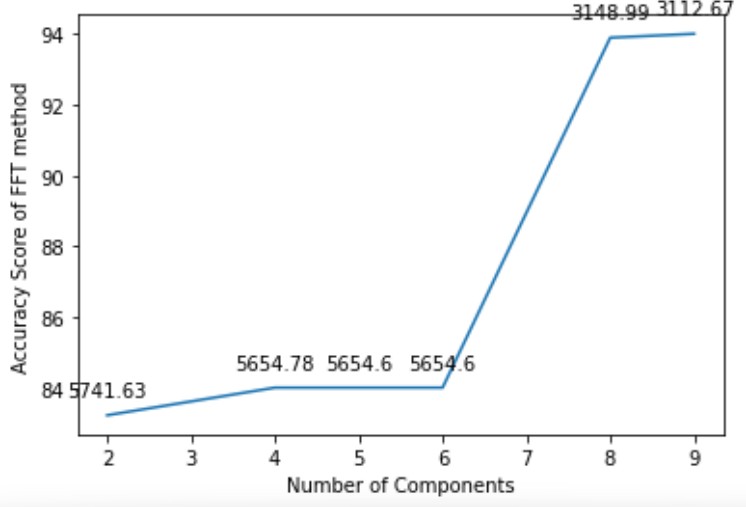
As we can observe after 100 components, the accuracy of PCA method increases by a huge margin giving more accurate results. The accuracy obtained after considering 300componentsis95.59%.

The values of x-axis are original value/30. Example, for 3, original value is 90 components.

# Using LDA built-in function from Linear Discriminant Library:

As we can observe the accuracy score changes by a huge amount after 5 components are considered. In LDA, we consider the number of components equal to or less than the number of classes available in the dataset. At 9 components, the maximum number of components that can be considered as we have 10 classes. We obtain an accuracy of 93.91% at maximum using LDA method of image compression.

# Using FFT – Fast Fourier Transformations from NumPy library:

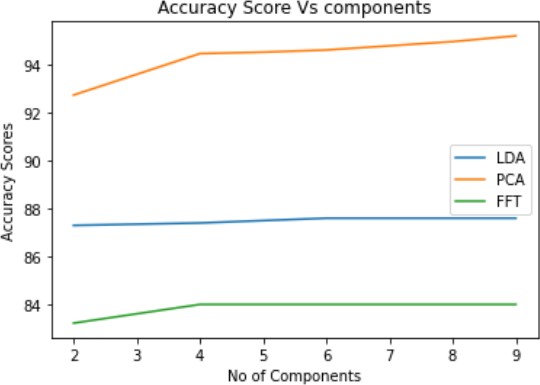


As we can observe this is not such a great technique of image compression. As the data of the image pixel is converted into complex numbers and using them reconstruction of images is not as accurate as other methods. The maximum accuracy that is obtained after considering 8 components is 94.01%.

This method is not widely used as it also takes a huge amount of time to compress the data. The time dependency will also be analyzed in the section 5.3.

The values of x-axis are original value/30. Example, for 3, original value is 90 components.

# Comparison between techniques based on number of components considered



(Manual=Eigen Value of Covariance)

As we can observe, all the techniques provide satisfactory result if we consider 240+components. But, for lower components, Eigen value of covariance and PCA are much better techniques. Still, PCA and Eigen value of covariance have their own drawbacks.

Since the data considered is linear, we observe that the two techniques, namely PCA, Eigen Value of covariance out-performs FFT. But, when it comes to nonlinear data, these techniques aren’t useful.

# 5.2: Analysis of training dataset size required for precise compression.

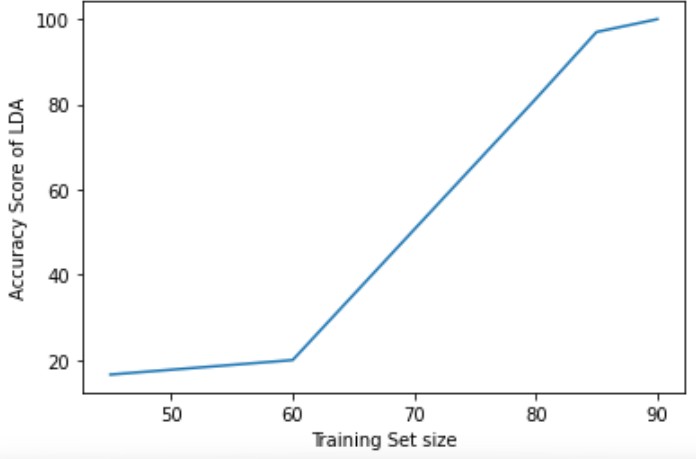
The amount of training dataset to be considered also matters for data compression as we can compress less amount of data but not get required results. On the other hand, we can compress more data which may consume more time and give very high results. It is also a necessity to find the right amount of data to be considered as the time required for compression shouldn’t be too long and result also must be at a satisfactory level. We are considering the IRIS dataset as we don’t have test dataset available for it, we will be using the dataset to construct both training and test datasets from the original dataset itself.

Various techniques Vs size of training set

# Using PCA method:

As we can observe that the required amount of dataset size for PCA method would be80%.This is in particular with IRIS dataset, but it can also be considered for other dataset. Accuracy touches 100% that is at 90% of dataset to be considered as training set size, but for training data to be 90%, compression would take more time. At 80% of original dataset as training set, we would obtain a accuracy of 93% approximately which is very good score compared to the time taken for compression of data.

# Using LDA method:

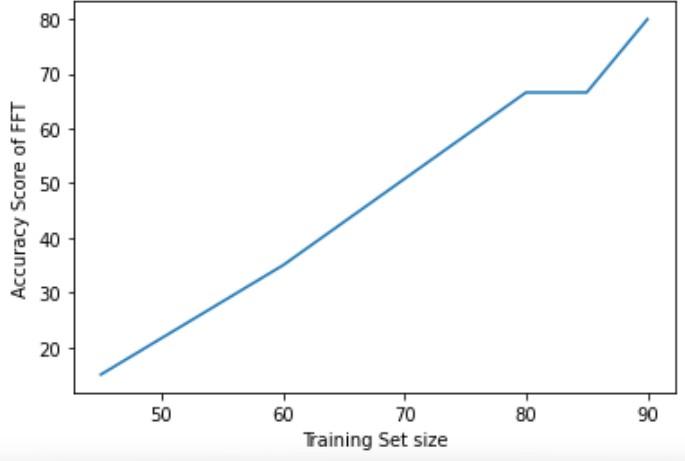


As we can observe, for LDA, the right amount of training set size would be around 85%of original dataset. If we consider the dataset to be less than 80%, we obtain very low accuracy and if we consider training dataset size of 90% of the original dataset, though we obtain a lot higher accuracy still the time required for data compression increases by a huge amount. At 80% of original dataset as training set, we would obtain a accuracy of 80% approximately which is satisfactory.

# Using Eigen value of covariance method: (Manual method = Eigen value of covariance method)

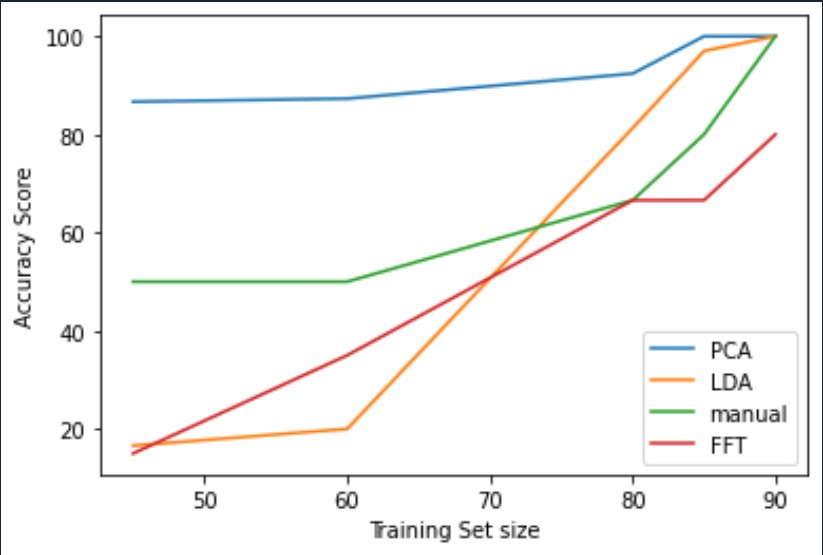
From the graph, it is clear that we need the training dataset size to be around 85% for this technique of compression. Going below 80% would reduce the accuracy and considering above 85% of original dataset for training set would increase the time needed for compression. As observed at 85%, we would obtain a accuracy of 85% approximately.

# Using FFT method of transformation after compression:



We can observe that there is constant increase in accuracy score using Fast Fourier Transformation method. As we can see, the accuracy score is below 80% even after considering 90% of original dataset as training set. So, deciding the optimum amount of dataset as training set comes down to the requirements of the programmer. Considering 90% of dataset as training set would lead to more time required for compression and lower percentage of dataset as training set would lead to low accurate results. So, it is the choice of the user to select the percentage of training set from the original dataset.

# Combined view of all Techniques Vs Training dataset size graph



* 1. **5.3: Analysis of Techniques with respect to the time taken to perform it:**

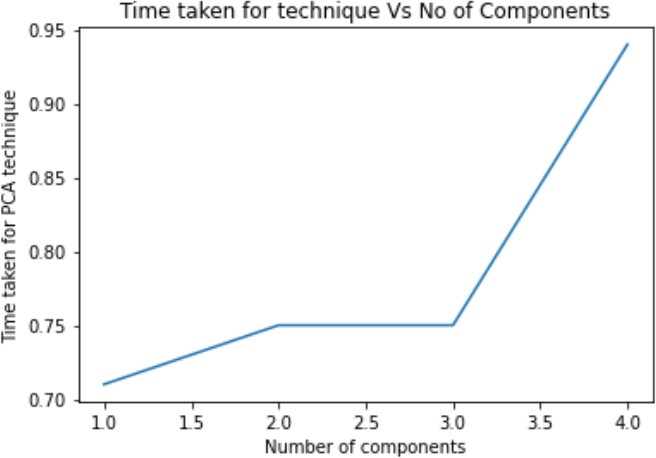
Considering the number of features available in the GENE-LEUKEMIA dataset, it appears that it would be suitable for analyzing the time taken by each technique for considering the key component or components for transforming the data or reducing the dimensions of the data.

The time taken by each technique varies a lot as there are 22283 features from the dataset. Considering the time to reduce the dimensions and then find the most similar data point from the training set to the new input data point.

Time taken by the technique = (Selecting the feature to be considered as key dimension) + (Transforming the data) + (Finding the most similar point from the training dataset for the given input data point)

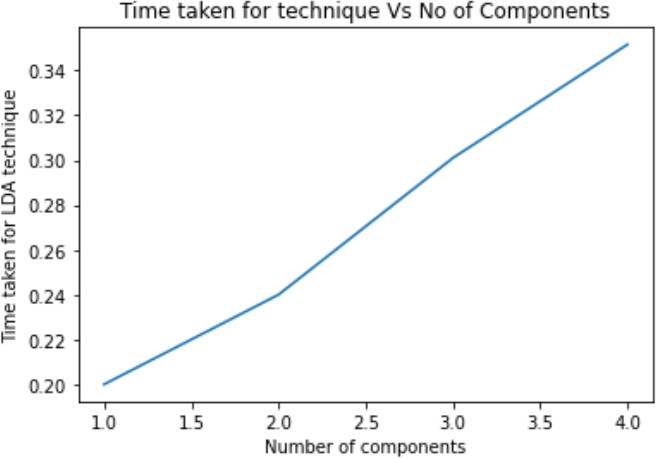
(All the time taken in the graph is considered in seconds)

# Using PCA method:



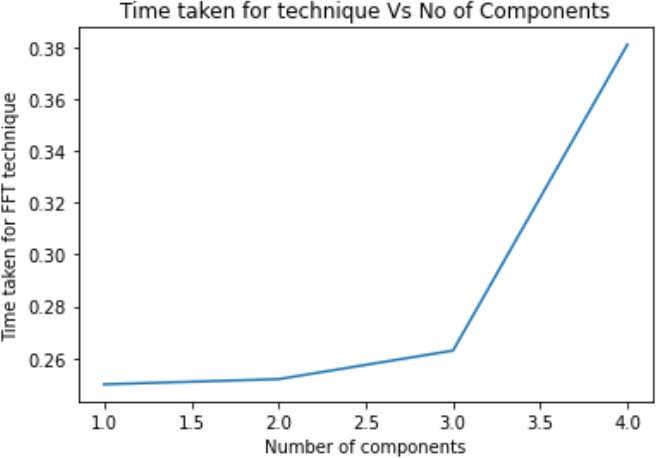
As it is visible in the graph, for PCA, as the number of components increase, time increases. Hence for the other number of components, the time taken is higher than expected.

# Using LDA method:



We can clearly observe that the time taken for higher components is high. As the number of components increases the time taken by the model to classify the data increases. As it has to consider the different types of data combinations from various features to classify the data points in the dataset, the time of execution increases. The optimal number of components would be 3 as for 1 component, the accuracy of PCA method is 87.5%, but for 3 components, it is 91.3% which is much better for the model. The number of components can also be decided based on the requirement, if time is a important parameter then, 1 component would be ideal, but if accuracy is the goal, then 3 components would be ideal.

# Using FFT method:



From the graph, it is visible that, when the number of components are 3, it takes the most time for transforming and then reconstructing the entire data using inverse transformation to find the most similar image. But, for 4 components, we can observe that the time taken is less and accuracy is 98% (on average) approximately. The accuracy is lowest for 1 component–approximately 87.56%.

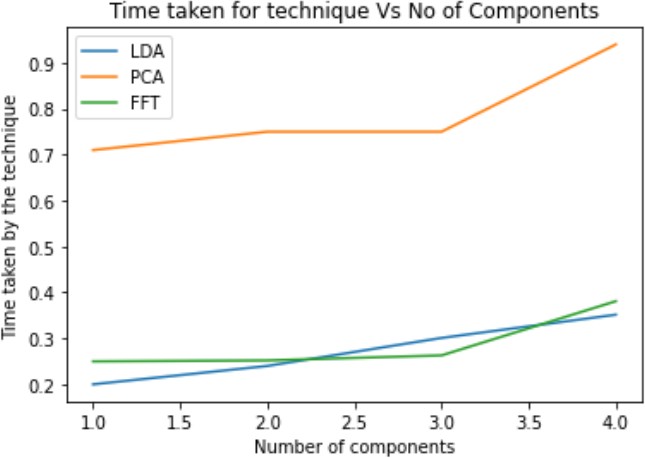
# Using Eigen value of Covariance method:

From the graph, it is clear that, computation time decreases as number of eigen vectors increase.

This decrease is observed because the time for reconstruction of data reduces and hence, the total time reduces. Also, the accuracy also keeps increasing as the number of components increases.

# Combined view for comparing techniques: [Techniques: PCA, LDA, FFT]

**Time taken Vs Number of components**

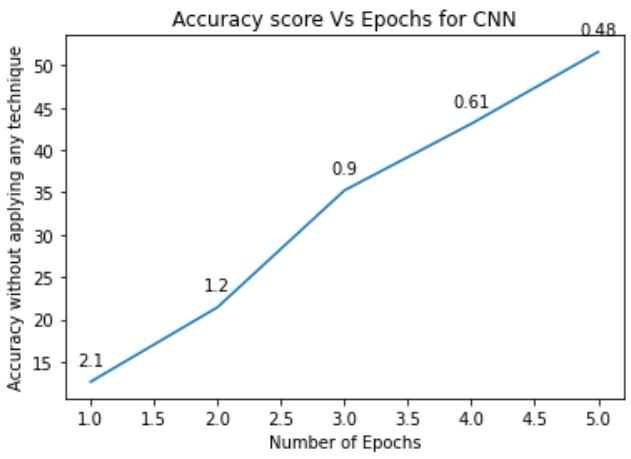


1. **Impact of Dimensionality reduction on Deep Learning Algorithms**

**Deep Learning Algorithms:**

There are several Deep Learning Algorithms used for predicting the data using different models  
  
For analysis, we will be using the most widely used CNN model for predicting the image labels   
  
for MNIST dataset. CNN – It is a type of neural network which used mainly because of easy   
  
training time.

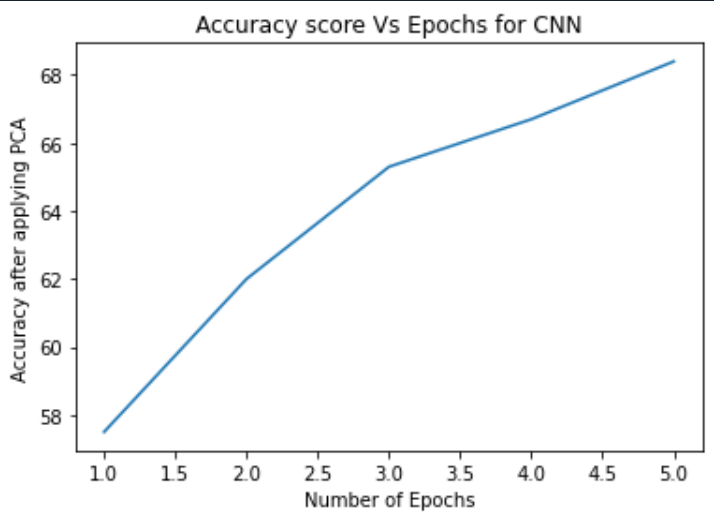
**Without applying any dimensionality reduction technique:**

Let’s see the accuracy obtained if any dimensionality reduction technique is not applied to the   
  
data before passing it to the CNN model.   
  
The graph depicts the Accuracy Score Vs Number of Epochs

The graph shows that the accuracy of the model increases which increasing number of epochs   
  
i.e. with increasing number of iterations, accuracy score increases of the neural network.  
  
The data present on the line of plot depicts the loss at each step. The loss needs to be multiplied with  
  
1000 to the original obtained value while execution. The values of accuracy and loss are average of   
  
10 whole executions of the program, giving more accurate results.  
  
As we increase the number of epochs, we can even achieve higher accuracies. For 50 epochs, we obtain  
  
a accuracy of 97.4%. But, for demonstration and understanding purpose, low epochs are considered.

**Impact of PCA on the CNN model:**

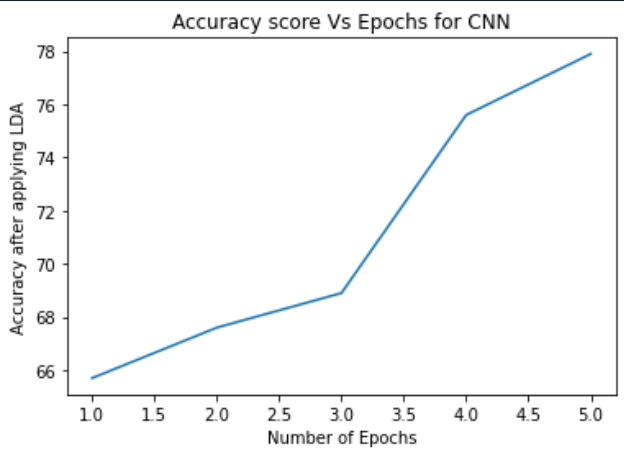
Using PCA, we compress the data using feature selection, which will be the input to the neural  
  
network.   
  
The accuracy score after applying PCA can be observed as follows:



As we can observe in the graph, the accuracy score of the model increases by a huge value  
  
as PCA removes the unnecessary data in the images which reduces the time of training  
  
the model as well as makes the model to focus on the important data and hence   
  
giving more accurate results. The components considered for PCA is 100 components.

**Impact of LDA on the CNN model:**

Using LDA, we compress the data using built in feature selection procedure, then the new formed  
  
data is passed to the Neural Network.   
  
The accuracy score after applying LDA can be observed as follows:



As we can observe from the graph, it is clear that the accuracy score obtained for each epoch   
  
is much greater than the accuracy obtained with original dataset, the number of components   
  
considered for this data is 5, which makes the accuracy better and required less time to train.  
  
The accuracy is even better than what is obtained using PCA method of compression and applying  
  
CNN.

1. **Let’s look at the different ways of transforming/representing data**

Different Spaces to represent data

* Eigen space
* Wavelet transform
* Discrete Fourier Transform
* Fast Fourier Transform
* Principle Component Analysis
* Singular Value Decomposition
* Discrete Cosine Transform
* Etc. There are many more ways to represent data, but, let’s look at the mentioned above.  
    
  Each technique has their own use and a specific projection and representation form. The   
  form of projection and transformation depend on specific mathematical operation involved   
  in it and the main goal of transformation

1. Eigen Value representation: In this form of representation, the data is considered in matrix form and then we find the eigenvectors and eigenvalues of the data. We use the eigen vectors to reconstruct the entire dataset. Hence, the space of representing the data consists of vectors. The vectors can be sorted according to the corresponding Eigen value for proper representation of data.
2. Wavelet Transform: In this transformation, the signal is decomposed into different frequency components at different scales using wavelet functions. The projected data is obtained by representing the signal as a linear combination of wavelet coefficients at different scales and positions.
3. Discrete Fourier Transform: DFT transforms a signal into it’s frequency components, similar to the Fourier Transform, but, using complex numbers. The projected data is represented by the DFT coefficients, which captures the energy distribution in the signal. It is the naïve version of Fast Fourier Transform and advanced version of Fourier transform.
4. Fast Fourier Transform: Fast Fourier transform is a transformation in which signal is decomposed into a sum of complex exponential functions with different frequencies. The projected data is represented by complex numbers in either Amplitude or phase and even in Amplitude and phase.
5. Principle Component Analysis: PCA projects the data onto a new coordinate system defined by using the principle component as axes for the new coordinate system. Both the axes are orthogonal to each other representing the directions of maximum variance in the data. The data is a linear combination of the principal components.
6. Singular Value Decomposition: Singular Value Decomposition or SVD decomposes a matrix into three matrics: U, V and ∑, where ∑ contains singular values that capture the importance of each vector. The projected data is obtained by selecting a subset of singular values and corresponding basis vectors from the decomposition.
7. Discrete Cosine Transform: DCT or Discrete Cosine Transform method transforms a signal into it’s frequency components similar to Fourier transforms, but the data is in real numbers here. The projected data is represented by the DCT coefficients, which capture the energy distribution in the signal and hence the space of data is created.

# Future Applications:

The uses of dimensionality reduction

Dimensionality reduction is an unsupervised machine learning technique that can be applied to your input data, without having the class column. If the data has 10 features, then the dimensions that can be reduced are 10. It means that your data can be plotted in a 10-dimensional space. This is not a practical approach in the present world. Hence, we use dimensionality reduction.

* + Visualization of data can be achieved through dimensionality reduction, which is also a key component of EDA- Exploratory Data Analysis. Hence, dimensionality reduction is like a prior step for visualization of practical data- i.e., data with a very high number of features.
  + The problem of over fitting the data can be overcome if dimensionality reduction is applied to the data given. It is one of the worst problem faced in data analysis and also while building models. Hence, reducing the dimensions of data will overcome this problem and produce more accurate results for testing inputs.
  + Dimensionality reduction also deals with one of the real-world issue i.e. it helps in training the model quicker as we reduce the total features that are required for building the model. Therefore huge algorithms which require greater time can be done quicker with the help of dimension reduction.
  + It also helps in converting nonlinear data into linearly-separable data form.
  + LDA, PCA, Eigen Value of covariance method, FFT are few examples of dimension-ality reduction and data compression as we observed for the mentioned datasets.

# References

* Tutorial on Principal Component Analysis Version2 by Jonathon Shlens (Referredforpage2–PCA)
* What is LDA?:Linear Discriminant Analysis for Machine Learning Author: Priyankur Sarkar, Published on:27thDec2022

(Referred for page 3,4– LDA explanation)

* Article and Report on Fast Fourier Transformation-FFT basics by NTi (Referredforpage4–FFT explanation)
* Eigen faces for Face Detection/Recognition

(M.Turkand & A.Pentland, *Journal of Cognitive Neuroscience*, vol.3) (Referred for page 5 – Eigen value of covariance method)

* Historical Notes of data compression from Stephen Wolfram Chapter 10, Processes of Perception and Analysis, section: Data Compression

(Referredforpage8–History/related work)

* Uses of Dimensionality Reduction by Rukshan Pramoditha published in Towards data Science on Dec 8, 2021

(Referred for page 25 – Future Applications)

* Mathematical Methods for Data Visualization by San José University   
  Author – Dr. Guangliang Chen

( Referred on Page-5 :Calculations related to LDA)

# Using PCA to Reduce Number of Parameters in a Neural Network by 30x Times. Published by Rukshan Pramodita in TowrdsDataScience

(Referred in page 35 – Effect of PCA on CNN model)

# Python Code for applying mentioned dimensionality reduction techniques

**And using it to find the most similar image from the dataset for the given image input:**

#importing all the libraries import numpy as np

From scipy.spatial.distance import cdist

From sklearn.metrics import accuracy\_score,confusion\_matrix   
import matplotlib.pyplot as plt

#Forming the training images from 1st dataset with open("<1stDataset>",'rb') as f:

data=np.fromfile(f,dtype=np.uint8)

training=data[16:]

train\_output=training.reshape((60000,28\*28))

X\_train=training.reshape((60000,28\*28))

#Using 2nd dataset label the digits in training sets, split the 1st dataset for each digit

withopen("<2nddataset>","rb")asf:

train\_label = np.fromfile(f,dtype = np.uint8)Y\_train= train\_label[8:60008]

#Forming the testing images from 3rd dataset withopen("<3rddataset>","rb") as f:

test=np.fromfile(f,dtype=np.uint8)  
test\_set=test[16:7840017]

X\_test=test\_set.reshape((10000,28\*28))

#Building confusion matrix for both the methods using the 4th dataset. withopen("<4thdataset>","rb")asf:

test\_label = np.fromfile(f,dtype = np.uint8)Y\_test =test\_label[8:10009]

pred\_test=[]

pred\_test\_PCA=[]

pred\_test\_FFT=[]

train0 = []  
train1 = []  
train2 = []  
train3 = []  
train4 = []  
train5 = []  
train6 = []  
train7 = []  
train8 = []  
train9=[]

for i in range(len(Y\_train)):

ifY\_train[i]==0:

train0.append(X\_train[i])

elifY\_train[i]==1:

train1.append(X\_train[i])

elifY\_train[i]==2:

train2.append(X\_train[i])

elifY\_train[i]==3:

train3.append(X\_train[i])

elifY\_train[i]==4:

train4.append(X\_train[i])

elifY\_train[i]==5:

train5.append(X\_train[i])

elifY\_train[i]==6:

train6.append(X\_train[i])

elifY\_train[i]==7:

train7.append(X\_train[i])

elifY\_train[i]==8:

train8.append(X\_train[i])

elifY\_train[i]==9:

train9.append(X\_train[i])

mean\_image0 = np.mean(train0,axis = 0)  
 mean\_image1 = np.mean(train1,axis = 0)

mean\_image2 = np.mean(train2,axis = 0)

mean\_image4 = np.mean(train4,axis = 0)

mean\_image3 = np.mean(train3,axis = 0)

mean\_image5 = np.mean(train5,axis = 0)

mean\_image6 = np.mean(train6,axis = 0)

mean\_image7 = np.mean(train7,axis = 0)

mean\_image8 = np.mean(train8,axis = 0)

mean\_image9 = np.mean(train9,axis = 0)

n = n\_PCA = n\_LDA = 50

#EIGENVALUESOFCOVARIANCE method

data\_center0=train0-mean\_image0  
#finding the covariance matrix

cov0 = np.cov(data\_center0,rowvar = False)

eigenvalues0,eigenvectors0=np.linalg.eigh(cov0)

sort\_index0 = np.argsort(eigenvalues0)[::-1]

eigenvalues0 = eigenvalues0[sort\_index0]

eigenvectors0=eigenvectors0[:,sort\_index0]

projected\_data0=np.dot(data\_center0,eigenvectors0[:,:n])

data\_center1=train1-mean\_image1

cov1 = np.cov(data\_center1,rowvar = False)  
eigenvalues1,eigenvectors1=np.linalg.eigh(cov1)  
sort\_index1 = np.argsort(eigenvalues1)[::-1]  
eigenvalues1 = eigenvalues1[sort\_index1]  
eigenvectors1=eigenvectors1[:,sort\_index1]

projected\_data1=np.dot(data\_center1,eigenvectors1[:,:n])  
data\_center2=train2-mean\_image2

cov2=np.cov(data\_center2,rowvar=False)

eigenvalues2,eigenvectors2=np.linalg.eigh(cov2)

sort\_index2 = np.argsort(eigenvalues2)[::-1]

eigenvalues2 = eigenvalues2[sort\_index2]

eigenvectors2=eigenvectors2[:,sort\_index2]

projected\_data2=np.dot(data\_center2,eigenvectors2[:,:n])

data\_center3=train3-mean\_image3

cov3 = np.cov(data\_center3,rowvar = False)  
eigenvalues3,eigenvectors3=np.linalg.eigh(cov3)  
sort\_index3 = np.argsort(eigenvalues3)[::-1]  
eigenvalues3 = eigenvalues3[sort\_index3]  
eigenvectors3=eigenvectors3[:,sort\_index3]

projected\_data3=np.dot(data\_center3,eigenvectors3[:,:n])  
data\_center4=train4-mean\_image4

cov4 = np.cov(data\_center4,rowvar = False)  
eigenvalues4,eigenvectors4=np.linalg.eigh(cov4)  
sort\_index4 = np.argsort(eigenvalues4)[::-1]  
eigenvalues4 = eigenvalues4[sort\_index4]  
eigenvectors4=eigenvectors4[:,sort\_index4]

projected\_data4=np.dot(data\_center4,eigenvectors4[:,:n])  
data\_center5=train5-mean\_image5

cov5 = np.cov(data\_center5,rowvar = False)  
eigenvalues5,eigenvectors5=np.linalg.eigh(cov5)  
sort\_index5 = np.argsort(eigenvalues5)[::-1]  
eigenvalues5 = eigenvalues5[sort\_index5]  
eigenvectors5=eigenvectors5[:,sort\_index5]

projected\_data5=np.dot(data\_center5,eigenvectors5[:,:n])  
data\_center6=train6-mean\_image6

cov6 = np.cov(data\_center6,rowvar = False)  
eigenvalues6,eigenvectors6=np.linalg.eigh(cov6)  
sort\_index6 = np.argsort(eigenvalues6)[::-1]  
eigenvalues6 = eigenvalues6[sort\_index6]  
eigenvectors6=eigenvectors6[:,sort\_index6]

projected\_data6=np.dot(data\_center6,eigenvectors6[:,:n])  
data\_center7=train7-mean\_image7

cov7 = np.cov(data\_center7,rowvar = False)  
eigenvalues7,eigenvectors7=np.linalg.eigh(cov7)  
sort\_index7 = np.argsort(eigenvalues7)[::-1]  
eigenvalues7 = eigenvalues7[sort\_index7]  
eigenvectors7=eigenvectors7[:,sort\_index7]

projected\_data7=np.dot(data\_center7,eigenvectors7[:,:n])  
data\_center8=train8-mean\_image8

cov8 = np.cov(data\_center8,rowvar = False)  
eigenvalues8,eigenvectors8=np.linalg.eigh(cov8)  
sort\_index8 = np.argsort(eigenvalues8)[::-1]  
eigenvalues8 = eigenvalues8[sort\_index8]  
eigenvectors8=eigenvectors8[:,sort\_index8]

projected\_data8=np.dot(data\_center8,eigenvectors8[:,:n])  
data\_center9=train9-mean\_image9

cov9 = np.cov(data\_center9,rowvar = False)  
eigenvalues9,eigenvectors9=np.linalg.eigh(cov9)  
sort\_index9 = np.argsort(eigenvalues9)[::-1]  
eigenvalues9 = eigenvalues9[sort\_index9]  
eigenvectors9=eigenvectors9[:,sort\_index9]

projected\_data9=np.dot(data\_center9,eigenvectors9[:,:n])  
#finding the new projection of data

for i in X\_test:

input\_image = i

new\_center0=input\_image-mean\_image0new\_projection0=np.dot(new\_center0,eigenvectors0[:,:n])

dist0=int(min(min(cdist(new\_projection0.reshape((1,1)),projected\_data0,'euclidean'))))  
new\_center1=input\_image-mean\_image1

new\_projection1=np.dot(new\_center1,eigenvectors1[:,:n])

dist1=int(min(min(cdist(new\_projection1.reshape((1,-1)),projected\_data1,'euclidean'))))  
new\_center2=input\_image-mean\_image2

new\_projection2=np.dot(new\_center2,eigenvectors2[:,:n])

dist2=int(min(min(cdist(new\_projection2.reshape((1,-1)),projected\_data2,'euclidean'))))  
new\_center3=input\_image-mean\_image3

new\_projection3=np.dot(new\_center3,eigenvectors3[:,:n])

dist3=int(min(min(cdist(new\_projection3.reshape((1,-1)),projected\_data3,'euclidean'))))  
new\_center4=input\_image-mean\_image4

new\_projection4=np.dot(new\_center4,eigenvectors4[:,:n])

dist4=int(min(min(cdist(new\_projection4.reshape((1,-1)),projected\_data4,'euclidean'))))  
new\_center5=input\_image-mean\_image5

new\_projection5=np.dot(new\_center5,eigenvectors5[:,:n])

dist5=int(min(min(cdist(new\_projection5.reshape((1,-1)),projected\_data5,'euclidean'))))  
new\_center6=input\_image-mean\_image6

new\_projection6=np.dot(new\_center6,eigenvectors6[:,:n])

dist6=int(min(min(cdist(new\_projection6.reshape((1,-1)),projected\_data6,'euclidean'))))  
new\_center7=input\_image-mean\_image7

new\_projection7=np.dot(new\_center7,eigenvectors7[:,:n])

dist7=int(min(min(cdist(new\_projection7.reshape((1,-1)),projected\_data7,'euclidean'))))  
new\_center8=input\_image-mean\_image8

new\_projection8=np.dot(new\_center8,eigenvectors8[:,:n])

dist8=int(min(min(cdist(new\_projection8.reshape((1,-1)),projected\_data8,'euclidean'))))  
new\_center9=input\_image-mean\_image9

new\_projection9=np.dot(new\_center9,eigenvectors9[:,:n])

dist9=int(min(min(cdist(new\_projection9.reshape((1,-1)),projected\_data9,'euclidean'))))  
dist=[dist0,dist1,dist2,dist3,dist4,dist5,dist6,dist7,dist8,dist9]

index=dist.index(min(dist))pred\_test.append(index)

print("Accuracy Score of manual method is:", accuracy\_score(pred\_test,Y\_test)\*100,"%")  
Conf\_matrix\_manual = np.array(confusion\_matrix(pred\_test,Y\_test))  
print("Correctpredictions:",np.trace(Conf\_matrix\_manual))

print("Wrong predictions:",np.sum(Conf\_matrix\_manual)-np.sum(np.trace(Conf\_matrix\_manual)))  
#BUILT IN FUNCTION PCA method

from sklearn.decomposition import PCA p=PCA(n\_components=n\_PCA)

p.fit(train0)

#Applying compression mean\_image0=np.mean(train0)

projected\_data\_PCA0=p.transform(train0)  
mean\_image1 = np.mean(train1)  
projected\_data\_PCA1=p.transform(train1)  
mean\_image2 = np.mean(train2)  
projected\_data\_PCA2=p.transform(train2)  
mean\_image3 = np.mean(train3)  
projected\_data\_PCA3=p.transform(train3)  
mean\_image4 = np.mean(train4)  
projected\_data\_PCA4=p.transform(train4)  
mean\_image5 = np.mean(train5)  
projected\_data\_PCA5=p.transform(train5)  
mean\_image6 = np.mean(train6)  
projected\_data\_PCA6=p.transform(train6)  
mean\_image7 = np.mean(train7)  
projected\_data\_PCA7=p.transform(train7)  
mean\_image8 = np.mean(train8)  
projected\_data\_PCA8=p.transform(train8)

mean\_image9 = np.mean(train9)  
projected\_data\_PCA9=p.transform(train9)  
for i in X\_test:

input\_image=i

new\_image\_center\_PCA0 = input\_image - mean\_image0  
new\_projected\_PCA0=p.transform(new\_image\_center\_PCA0.reshape((1,-1)))

dist\_PCA0=int(min(min(cdist(new\_projected\_PCA0,projected\_data\_PCA0,'euclidean'))))  
new\_image\_center\_PCA1=input\_image-mean\_image1

new\_projected\_PCA1=p.transform(new\_image\_center\_PCA1.reshape((1,-1)))  
dist\_PCA1=int(min(min(cdist(new\_projected\_PCA1,projected\_data\_PCA1,'euclidean'))))  
new\_image\_center\_PCA2=input\_image-mean\_image2

new\_projected\_PCA2=p.transform(new\_image\_center\_PCA2.reshape((1,-1)))  
dist\_PCA2=int(min(min(cdist(new\_projected\_PCA2,projected\_data\_PCA2,'euclidean'))))  
new\_image\_center\_PCA3=input\_image-mean\_image3

new\_projected\_PCA3=p.transform(new\_image\_center\_PCA3.reshape((1,-1)))  
dist\_PCA3=int(min(min(cdist(new\_projected\_PCA3,projected\_data\_PCA3,'euclidean'))))  
new\_image\_center\_PCA4=input\_image-mean\_image4

new\_projected\_PCA4=p.transform(new\_image\_center\_PCA4.reshape((1,-1)))  
dist\_PCA4=int(min(min(cdist(new\_projected\_PCA4,projected\_data\_PCA4,'euclidean'))))  
new\_image\_center\_PCA5=input\_image-mean\_image5

new\_projected\_PCA5=p.transform(new\_image\_center\_PCA5.reshape((1,-1)))  
dist\_PCA5=int(min(min(cdist(new\_projected\_PCA5,projected\_data\_PCA5,'euclidean'))))  
new\_image\_center\_PCA6=input\_image-mean\_image6

new\_projected\_PCA6=p.transform(new\_image\_center\_PCA6.reshape((1,-1)))  
dist\_PCA6=int(min(min(cdist(new\_projected\_PCA6,projected\_data\_PCA6,'euclidean'))))  
new\_image\_center\_PCA7=input\_image-mean\_image7

new\_projected\_PCA7=p.transform(new\_image\_center\_PCA7.reshape((1,-1)))  
dist\_PCA7=int(min(min(cdist(new\_projected\_PCA7,projected\_data\_PCA7,'euclidean'))))  
new\_image\_center\_PCA8=input\_image-mean\_image8

new\_projected\_PCA8 = p.transform(new\_image\_center\_PCA8.reshape((1,-1)))  
dist\_PCA8=int(min(min(cdist(new\_projected\_PCA8,projected\_data\_PCA8,'euclidean')))

new\_image\_center\_PCA9 = input\_image - mean\_image9

new\_projected\_PCA9=p.transform(new\_image\_center\_PCA9.reshape((1,-1)))

dist\_PCA9 =int(min(min(cdist(new\_projected\_PCA9,projected\_data\_PCA9,'euclidean'))))

dist\_PCA=[dist\_PCA0,dist\_PCA1,dist\_PCA2,dist\_PCA3,dist\_PCA4,dist\_PCA5,dist\_PCA6,dist\_PCA7,dist\_PCA8,dist\_PCA9]  
#Finding the Least distance image from input\_image

index\_PCA=dist\_PCA.index(min(dist\_PCA))

pred\_test\_PCA.append(index\_PCA)

print("AccuracyScoreusingPCAis:",accuracy\_score(pred\_test\_PCA,Y\_test)\*100,"%")Conf\_matrix\_PCA=confusion\_matrix(pred\_test\_PCA,Y\_test)

print("Correctpredictions:",np.trace(Conf\_matrix\_PCA))

print("Wrong predictions:",np.sum(Conf\_matrix\_PCA)-np.sum(np.trace(Conf\_matrix\_PCA)))#BUILTINFunctionLDAmethod

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA lda=LDA(solver="svd",n\_components=9)

model =lda.fit(X\_train,Y\_train)

pred\_test\_LDA=model.predict(X\_test)

print("AccuracyscoreusingLDAmethodis",accuracy\_score(pred\_test\_LDA,Y\_test)\*100,"%")Conf\_matrix\_LDA=confusion\_matrix(pred\_test\_LDA,Y\_test)

print("Correctpredictions:",np.trace(Conf\_matrix\_LDA))

print("Wrongpredictions:",np.sum(Conf\_matrix\_LDA)-np.sum(np.trace(Conf\_matrix\_LDA)))#BUILTINFUNCTIONFFTmethod

from numpy.fft import fft2fromnumpy.fftimportifft2

new\_data\_FFT0 = abs(ifft2(fft2(train0)))

new\_data\_FFT1 = abs(ifft2(fft2(train1)))

new\_data\_FFT2 = abs(ifft2(fft2(train2)))

new\_data\_FFT3 = abs(ifft2(fft2(train3)))

new\_data\_FFT4 = abs(ifft2(fft2(train4)))

new\_data\_FFT5 = abs(ifft2(fft2(train5)))

new\_data\_FFT6 = abs(ifft2(fft2(train6)))

new\_data\_FFT7 = abs(ifft2(fft2(train7))))

new\_data\_FFT8 = abs(ifft2(fft2(train8)))

new\_data\_FFT9 = abs(ifft2(fft2(train9)))

for i in X\_test:

input\_image = i

new\_image\_center\_FFT0=input\_image-mean\_image0

new\_projected\_FFT0 = abs(ifft2(fft2(new\_image\_center\_FFT0.reshape((1,-1)))))

dist\_FFT0 = int(min(min(cdist(new\_projected\_FFT0,new\_data\_FFT0,'euclidean'))))  
new\_image\_center\_FFT1=input\_image-mean\_image1

new\_projected\_FFT1 = abs(ifft2(fft2(new\_image\_center\_FFT1.reshape((1,-1)))))

dist\_FFT1 = int(min(min(cdist(new\_projected\_FFT1,new\_data\_FFT1,'euclidean')))

)new\_image\_center\_FFT2=input\_image-mean\_image2

new\_projected\_FFT2 = abs(ifft2(fft2(new\_image\_center\_FFT2.reshape((1,-1)))))

dist\_FFT2 = int(min(min(cdist(new\_projected\_FFT2,new\_data\_FFT2,'euclidean'))))

new\_image\_center\_FFT3=input\_image-mean\_image3

new\_projected\_FFT3 = abs(ifft2(fft2(new\_image\_center\_FFT3.reshape((1,-1)))))

dist\_FFT3 = int(min(min(cdist(new\_projected\_FFT3,new\_data\_FFT3,'euclidean'))))

new\_image\_center\_FFT4=input\_image-mean\_image4

new\_projected\_FFT4 = abs(ifft2(fft2(new\_image\_center\_FFT4.reshape((1,-1)))))

dist\_FFT4 = int(min(min(cdist(new\_projected\_FFT4,new\_data\_FFT4,'euclidean'))))

new\_image\_center\_FFT5=input\_image-mean\_image5

new\_projected\_FFT5 = abs(ifft2(fft2(new\_image\_center\_FFT5.reshape((1,-1)))))

dist\_FFT5 = int(min(min(cdist(new\_projected\_FFT5,new\_data\_FFT5,'euclidean'))))

new\_image\_center\_FFT6=input\_image-mean\_image6

new\_projected\_FFT6 = abs(ifft2(fft2(new\_image\_center\_FFT6.reshape((1,-1)))))

dist\_FFT6 = int(min(min(cdist(new\_projected\_FFT6,new\_data\_FFT6,'euclidean'))))

new\_image\_center\_FFT7=input\_image-mean\_image7

new\_projected\_FFT7 = abs(ifft2(fft2(new\_image\_center\_FFT7.reshape((1,-1)))))

dist\_FFT7 = int(min(min(cdist(new\_projected\_FFT7,new\_data\_FFT7,'euclidean'))))

new\_image\_center\_FFT8=input\_image-mean\_image8

new\_projected\_FFT8 = abs(ifft2(fft2(new\_image\_center\_FFT8.reshape((1,-1)))))

dist\_FFT8=int(min(min(cdist(new\_projected\_FFT8,new\_data\_FFT8,'euclidean'))))

new\_image\_center\_FFT9=input\_image-mean\_image9

new\_projected\_FFT9 = abs(ifft2(fft2(new\_image\_center\_FFT9.reshape((1,-1)))))

dist\_FFT9=int(min(min(cdist(new\_projected\_FFT9,new\_data\_FFT9,'euclidean'))))

dist\_FFT=[dist\_FFT0,dist\_FFT1,dist\_FFT2,dist\_FFT3,dist\_FFT4,dist\_FFT5,dist\_FFT6,dist\_FFT7,dist\_FFT8,dist\_FFT9]

#Finding the Least distance image from input\_image

index\_FFT=dist\_FFT.index(min(dist\_FFT))  
pred\_test\_FFT.append(index\_FFT)

print("AccuracyScoreusingFFTis:",accuracy\_score(pred\_test\_FFT,Y\_test)\*100,"%")

Conf\_matrix\_FFT=confusion\_matrix(pred\_test\_FFT,Y\_test)

print("Correctpredictions:",np.trace(Conf\_matrix\_FFT))

print("Wrongpredictions:",np.sum(Conf\_matrix\_FFT)-np.sum(np.trace(Conf\_matrix\_FFT)))