RAMAIAH INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated To VTU), Bangalore

MACHINE LEARNING AND DEEP LEARNING (ECE642)

Mini-Project

Report On

Choosing a reduced dimension for the data using the Optdigits dataset and PCA, and calculation of PSNR for the test data comparing with original image.

Report By

SNEHA N - 1MS18EC113

INTRODUCTION

Principal components analysis (PCA) is a mapping from the original, high-dimensional space of 'd' input variables to a lower-dimensional subspace of 'k' variables. The aim is to reduce data loss as much as possible thereby maximising the amount of variance in the data explained by the principal components, while minimising the number of principal components.

In linear algebra terms, we are projecting an input vector X to a reduced input vector Z using a projection matrix W:

$$\mathbf{Z=W^TX}$$

The construction of the projection matrix is based on a maximization problem- maximizing the variance explained by each principal component. The principal component is W_1, a vector of length d, and is a weighted combination of input variables which, after the data is projected onto W_1, spreads the data out the most. The optimization problem thus becomes: for a given observation i, maximize the variance of the vector Z_i is equal to: $\mathbf{Var(Zi)=W^T_i\Sigma W_i}$
while constraining the L_2 norm of each projection vector to be 1: $\mathbf{W_i^T Wi=1}$

Peak Signal To Noise Ratio (PSNR) is commonly used to quantify reconstruction quality for images and video subject to lossy compression.

Problem Statement

Large datasets are increasingly common and are often difficult to interpret. In order to interpret such datasets, methods are required to drastically reduce their dimensionality in an interpretable way, such that most of the information in the data is preserved.

"Principal Component Analysis" (PCA) is one of the oldest and most widely used techniques. It works on a simple idea—reduce the dimensionality of a dataset, while preserving as much 'variability' (i.e. statistical information) as possible.
PCA reduces the dimensionality of large datasets and thereby increasing interpretability but at the same time minimizing information loss.

Handwritten Digits Data Set

The data set utilized is a handwritten digits data set provided by the UCI Machine Learning repository and consists of 8x8 optical scans of handwritten digits, together with labels of the correct digit.
Each observation corresponds to 8 by 8 grid indicating intensity. The intention is to map the 64 input variables (corresponding to pixel intensities on the 8x8 grid) to a categorical system response quantity (0 through 9). PCA does not make these types of mappings or predictions, however; PCA will project the inputs from 64-dimensional space onto a smaller space, thus reducing the cost of the problem and opening up more machine learning approaches.

CODE

#Importing Libraries

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import *
import seaborn as sns
import cv2
from pprint import pprint
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
```

#Loading the Dataset

```
testing_data = pd.read_csv(r'C:\Users\dell\Downloads\Optdigits\optdigits.tes',header=None)
X_testing,  y_testing  = testing_data.loc[:,0:63],  testing_data.loc[:,64]
low_memory=False
training_data = pd.read_csv(r'C:\Users\dell\Downloads\Optdigits\optdigits.tra',header=None)
X_training, y_training = training_data.loc[:,0:63], training_data.loc[:,64]
print(X_training.shape)
print(y_training.shape)
```

#Obtaining the 8 by 8 image *- The 64 inputs consist of an 8x8 image can be observed using #matplotlib's imshow() function*

```
original = X_training.loc[0,:].values.reshape(8,8)
print(original)
plt.imshow(original)
plt.title('Original image', fontsize=15, pad=15)
plt.savefig("original.png")
gca().grid(False)
plt.show()
```

```
[[ 0  1  6 15 12  1  0  0]
 [ 0  7 16  6  6 10  0  0]
 [ 0  8 16  2  0 11  2  0]
 [ 0  5 16  3  0  5  7  0]
 [ 0  7 13  3  0  8  7  0]
 [ 0  4 12  0  1 13  5  0]
 [ 0  0 14  9 15  9  0  0]
 [ 0  0  6 14  7  1  0  0]]
```

#Normalizing/Scaling the Data*- this function returns a standardized input matrix, mean and covariance matrix. This function will scale the variables so that they have a zero mean and a standard deviation of approximately 1.*

```
def get_normed_mean_cov(X):

    X_std = StandardScaler().fit_transform(X)
    X_mean = np.mean(X_std, axis=0)
    X_cov = (X_std - X_mean).T.dot((X_std - X_mean)) / (X_std.shape[0]-1)
    return X_std, X_mean, X_cov
X_std, X_mean, X_cov = get_normed_mean_cov(X_training)
X_std_validation, _, _ = get_normed_mean_cov(X_testing)
```

#Classification Task *- We create PCA and fit it to our standardized input data, then we transform our inputs into the reduced dimension space:*

```
pca2 = PCA(n_components=2, whiten=True)
pca2.fit(X_std)
X_red = pca2.transform(X_std)
```

#We create SVC(State Vector Classifier)and then pass the reduced dimensional data(64 to 2) to this state vector classifier and fit the reduced inputs to the desired system response.

```
linclass2 = SVC()
linclass2.fit(X_red,y_training)
```

#Subspace Projection: Visualizing Principal Components

#To color each point by the digit it represents, we create a color map with 10 elements (10 RGB values). Then, use the system response (y_training), which is a digit from 0 to 9.

```
def get_cmap(n):
    colorz = plt.get_cmap('Set1')
    return [ colorz(float(i)/n) for i in range(n)]
colorz = get_cmap(10)
colors = [colorz[yy] for yy in y_training]
scatter(X_red[:,0], X_red[:,1], color=colors, marker='*')
xlabel("Principal Component 0")
ylabel("Principal Component 1")
title("Handwritten Digit Data in\nPrincipal Component Space",size=14)
show()
```

#Analyzing the Data*-we visualize the* ***covariance matrix****. This tells us about the covariance between different pixels in the 8x8 images which means they are likely to be on opposite sides of the above average/below average scale.*

#Covariance= $Cov(\psi,\phi) = \sum_{i=1}^{N} \frac{(\varphi_i - \bar{\varphi})(\emptyset_i - \bar{\emptyset})}{N}$

```
fig = plt.figure(figsize=(12,12))
sns.heatmap(pd.DataFrame(X_cov), annot=False, cmap='PuOr')
```

```
#We visually see the structure of the 8x8 images. Images that are on opposite ends of the image have negative covariances.
plt.show()
```

#The problem of finding the principal components is an optimization problem. This optimization problem can be converted into a Lagrange multipliers problem, and solved by computing the eigenvalues and eigenvectors of the covariance matrix. The method of Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to equality constraints. Thus, the eigenvalues and eigenvectors of the covariance matrix yield the principal components.

#Eigenvectors and Eigenvalues of Covariance Matrix

```
eigenvals, eigenvecs = np.linalg.eig(X_cov)
print("Eigenvals shape: "+str(eigenvals.shape))
print("Eigenvecs shape: "+str(eigenvecs.shape))
```

```
# Creating a tuple(unsorted) of (eigenvalues, eigenvectors)
unsrt_eigenvalvec = [(np.abs(eigenvals[i]), eigenvecs[:,i]) for i in range(len(eigenvals))]
# Sorting tuple by eigenvalues
eigenvalvec = sorted(unsrt_eigenvalvec, reverse=True, key=lambda x:x[0])
```

#visualize the eigenvectors sorted by eigenvalue rank, and visually identify which eigenvector components dominate in each eigenvalue

```
fig = plt.figure(figsize=(14,3))
sns.heatmap(pd.DataFrame([pair[1] for pair in eigenvalvec[0:21]]), annot=False,
cmap='coolwarm', vmin=-0.5,vmax=0.5)
plt.ylabel("Ranked Eigenvalue")
plt.xlabel("Eigenvector Components")
plt.show()
```

#A list comprehension is looping over the first 21 eigenvalue-eigenvector pairs, which have been sorted above, and grabbing the eigenvector (pair[1]). This is put into a Data Frame and arranged for proper visualization. This is done mainly because we have no particularly meaningful labels for the 64 input components.

#Calculationof individual explained variances - λ_k *: The explained variance of the kth principal component (given d total principal components) is given by:*

$$\frac{\lambda_1+\lambda_2+\cdots+\lambda_k}{\lambda_1+\lambda_2+\cdots+\lambda_k+\cdots+\lambda_d}$$

#The distribution of explained variance for each principal component gives a sense of how much information will be represented and how much lost when the full 64-dimensional input is reduced using a principal component model.

```
lam_sum = sum(eigenvals)
explained_variance = [(lam_k/lam_sum) for lam_k in sorted(eigenvals, reverse=True)]
```

#Scree plotting *- scree plot is a line plot of the eigenvalues of factors or principal components in an analysis.*

```
plt.figure(figsize=(6, 4))
plt.bar(range(len(explained_variance)), explained_variance, alpha=0.5, align='center',
label='Individual Explained Variance $\lambda_{k}$')
plt.ylabel('Explained variance ratio')
plt.xlabel('Ranked Principal Components')
plt.title("Scree Graph")
plt.legend(loc='best')
plt.tight_layout()
```

#Cumulative explained variance-

```
fig = plt.figure(figsize=(6,4))
ax1 = fig.add_subplot(111)
ax1.plot(np.cumsum(explained_variance))
ax1.grid()
ax1.set_ylim([0,1.0])
ax1.set_xlabel('Number of Principal Components')
ax1.set_ylabel('Cumulative explained variance')
ax1.set_title('Explained Variance')
plt.show()
```

#Generating and saving the compressed image

```
pca_25 = PCA(n_components=25)
X_training_pca_25_reduced = pca_25.fit_transform(X_training)
X_training_pca_25_recovered = pca_25.inverse_transform(X_training_pca_25_reduced)
compressed_image = X_training_pca_25_recovered[1,:].reshape([8,8])
plt.imshow(compressed_image)
plt.title('Compressed image', fontsize=15, pad=15)
plt.savefig("compressed_image.png")
```

#Calculation of PSNR

```
img1=cv2.imread(r'C:\Users\dell\Desktop\Python\original.png')
img2=cv2.imread(r'C:\Users\dell\Desktop\Python\compressed_image.png')
psnr=cv2.PSNR(img1,img2)
print('The value of PSNR =',psnr,"dB")
```

CONCLUSION

Our project shows:

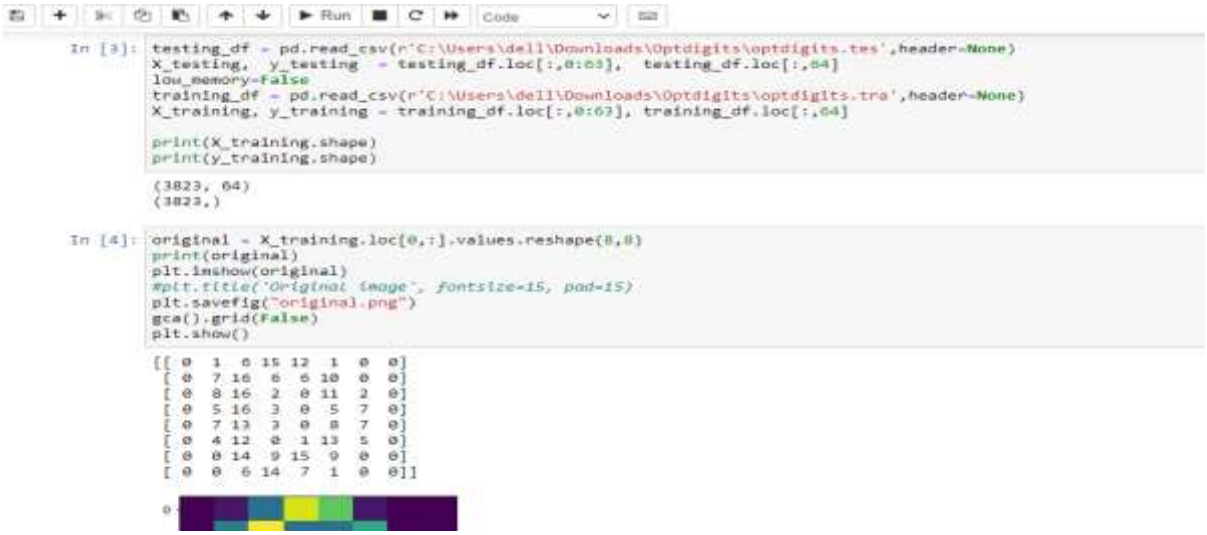
- Implementation details of a PCA model using handwritten digits dataset
- Visualization the Principal Components
- Calculation of PSNR

RESULTS

1. **90% of variance is explained within first 25 components.**
2. **PSNR = 19.3497 dB**

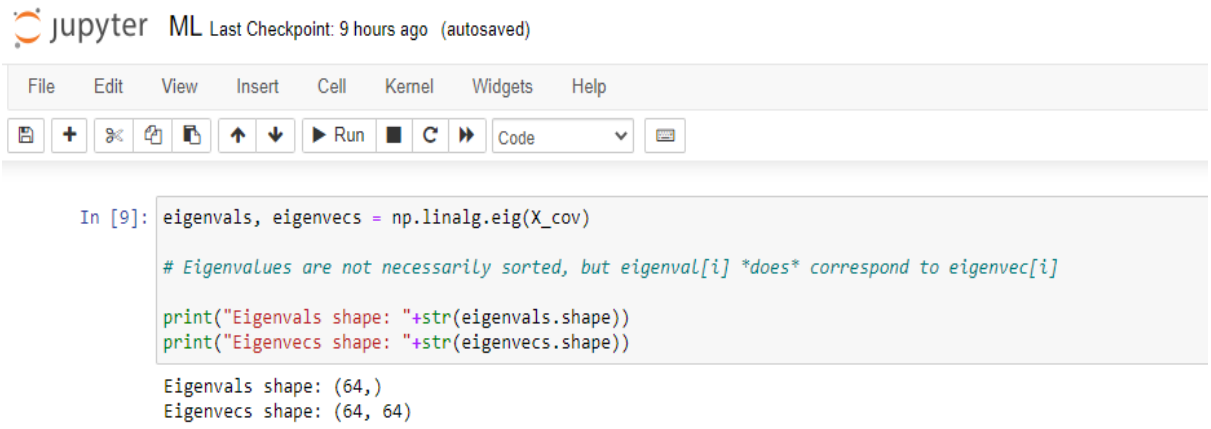
SCREENSHOTS

- **Checking the size of dataset and observing 8 x 8 matrix image of 64 inputs**



```
In [3]: testing_df = pd.read_csv(r'C:\Users\dell\Downloads\Optdigits\optdigits.tes',header=None)
        X_testing, y_testing = testing_df.loc[:,0:63], testing_df.loc[:,64]
        low_memory=False
        training_df = pd.read_csv(r'C:\Users\dell\Downloads\Optdigits\optdigits.tra',header=None)
        X_training, y_training = training_df.loc[:,0:63], training_df.loc[:,64]

        print(X_training.shape)
        print(y_training.shape)

(3823, 64)
(3823,)

In [4]: original = X_training.loc[0,:].values.reshape(8,8)
        print(original)
        plt.imshow(original)
        #plt.title('Original Image', fontsize=15, pad=15)
        plt.savefig("original.png")
        gca().grid(False)
        plt.show()

[[ 0  1  6 15 12  1  0  0]
 [ 0  7 16  6  6 10  0  0]
 [ 0  8 16  2  0 11  2  0]
 [ 0  5 16  3  0  5  7  0]
 [ 0  7 13  3  0  8  7  0]
 [ 0  4 12  0  1 13  5  0]
 [ 0  0 14  9 15  9  0  0]
 [ 0  0  6 14  7  1  0  0]]
```

- **Determining eigen values of Covariance matrix**



jupyter ML Last Checkpoint: 9 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

```
In [9]: eigenvals, eigenvecs = np.linalg.eig(X_cov)

        # Eigenvalues are not necessarily sorted, but eigenval[i] *does* correspond to eigenvec[i]

        print("Eigenvals shape: "+str(eigenvals.shape))
        print("Eigenvecs shape: "+str(eigenvecs.shape))

Eigenvals shape: (64,)
Eigenvecs shape: (64, 64)
```

➢ **Determination of PSNR**

jupyter ML Last Checkpoint: 13 hours ago (unsaved changes)

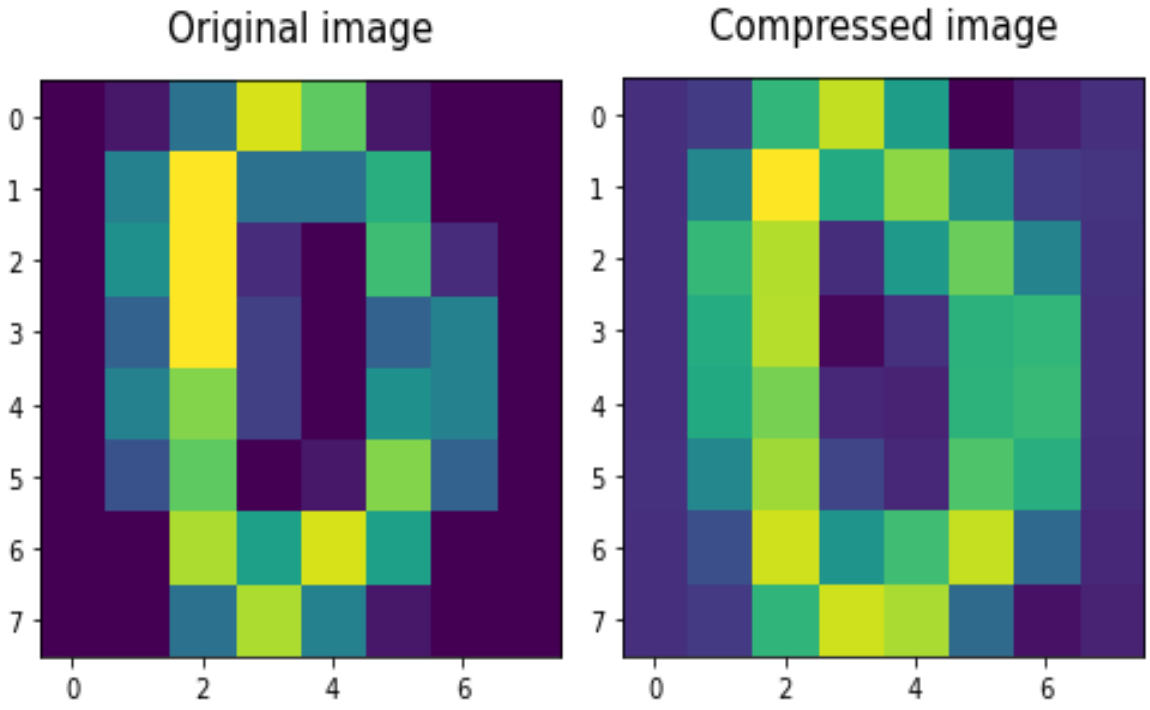
File Edit View Insert Cell Kernel Widgets Help

Run Code

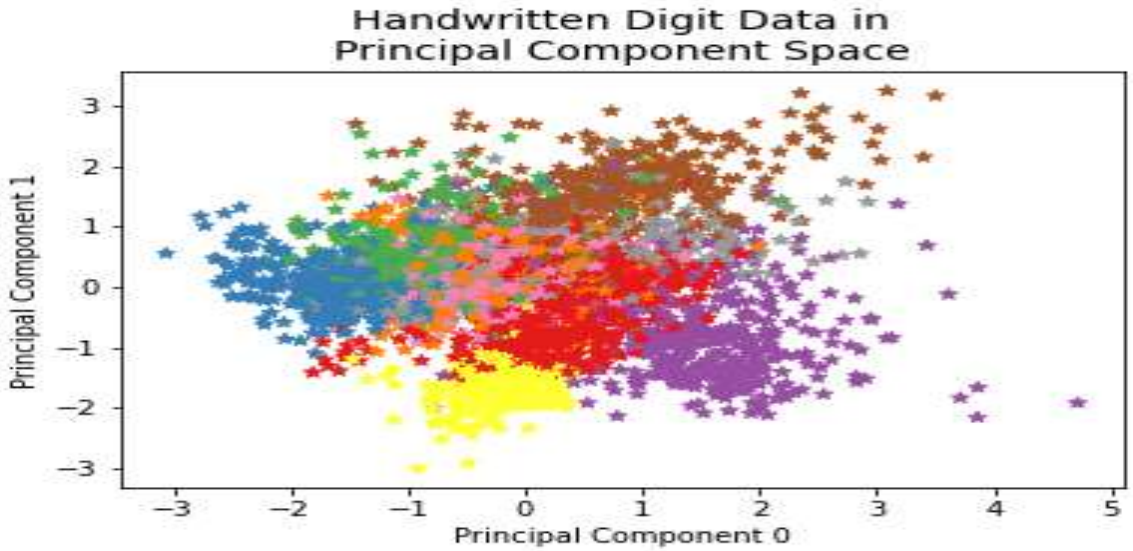
In [28]:
```
import cv2
img1=cv2.imread(r'C:\Users\dell\Desktop\Python\original.png')
img2=cv2.imread(r'C:\Users\dell\Desktop\Python\compressed_image.png')
psnr=cv2.PSNR(img1,img2)
print('The value of PSNR =',psnr,"dB")
```

```
The value of PSNR = 19.349677321550086 dB
```

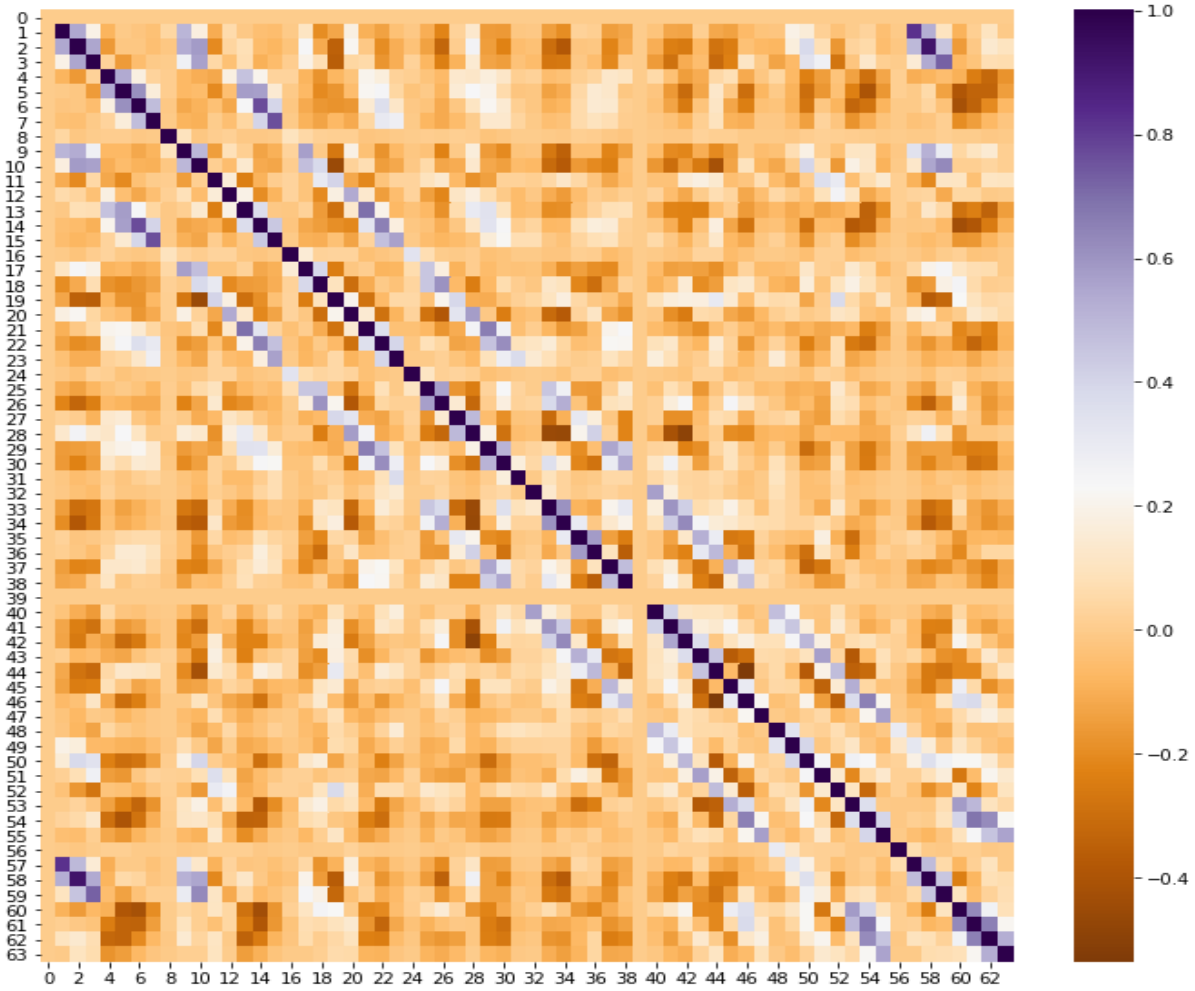
OUTPUTS



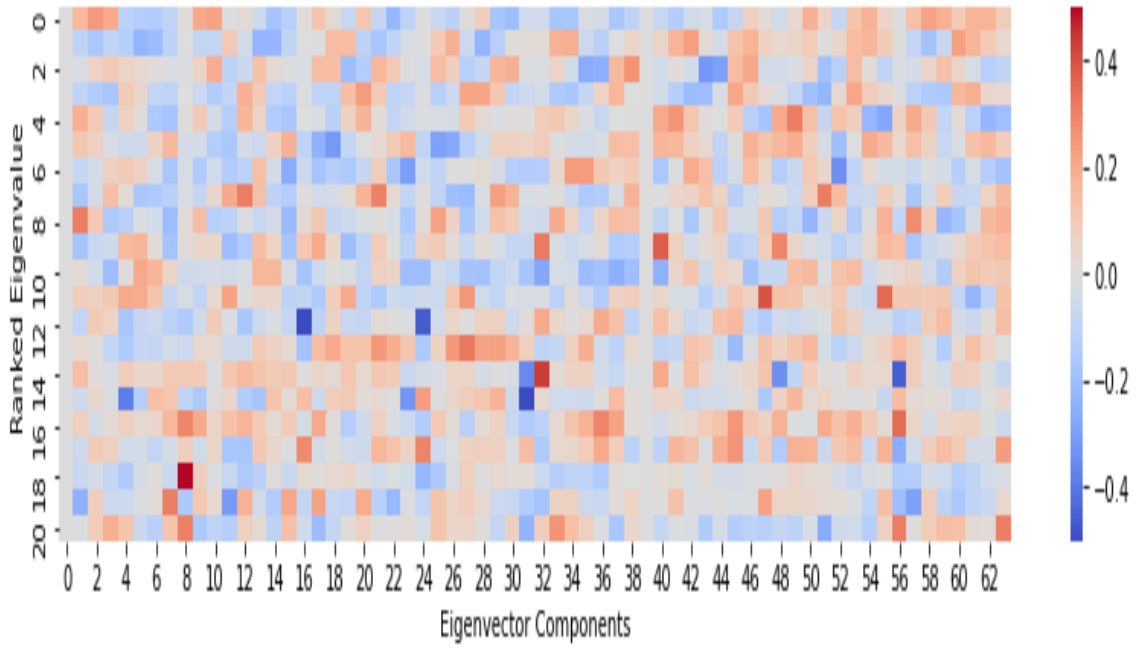
- **Visualizing Principal Components**



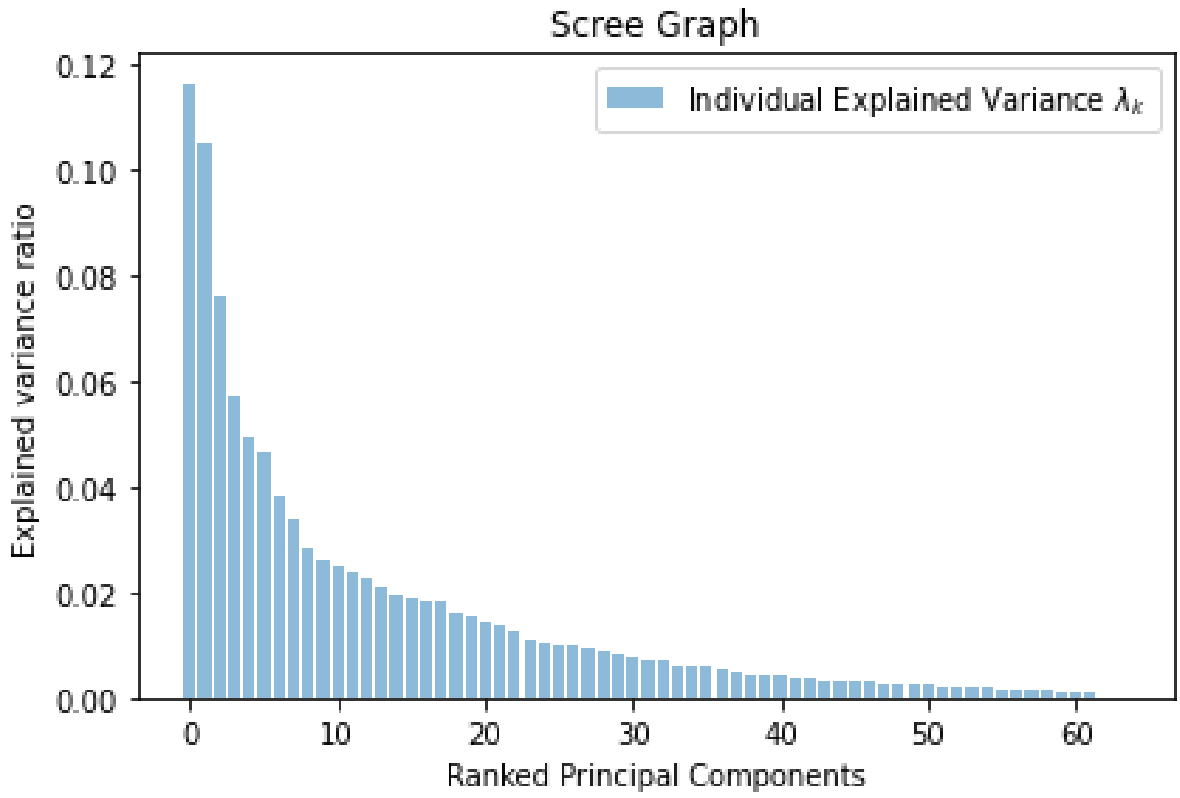
- **Visualize Covariance Matrix in the form of Heatmap** –This tells us about the covariance between different pixels in the 8x8 images.



- Heatmap of eigenvectors sorted by eigenvalue rank and visualise which eigenvector component dominates in each eigenvalue.



- **Scree Plot**



- **Cumulative Explained Variance** - Depicts 90% variance in the first 25 components.