HW2 Solution, CSE 569 Fall 2018

Name: Kunal Vinay Kumar Suthar
ASUrite ID: 1215112535

Question Attempting: Q (II)

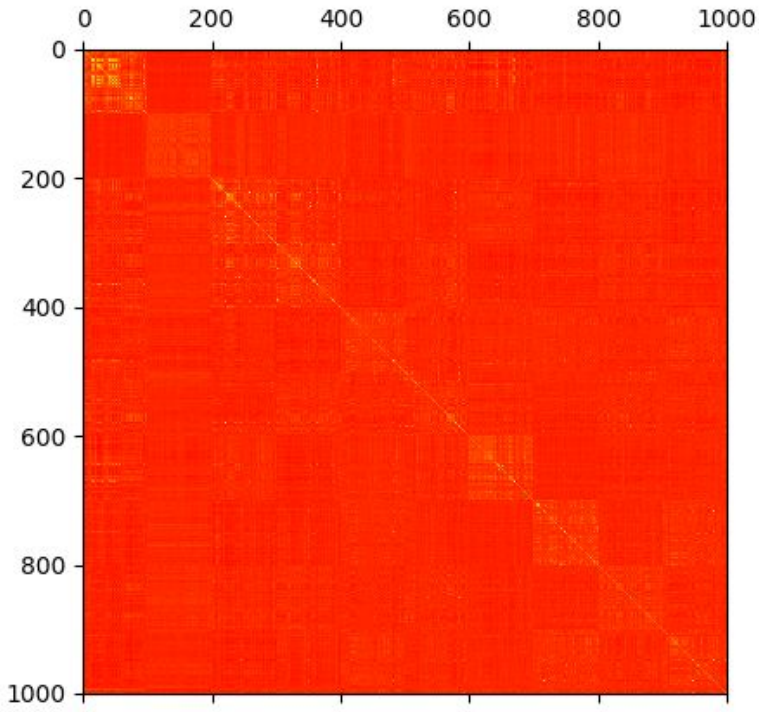
Q(II)

(a) Perform PCA and reduce the dimensions of training images from 28 × 28 = 784 to 100. Output the percentage of variance retained in the data after dimensionality reduction. Plot the covariance matrix for the training data after dimensionality reduction and analyze. Plot one example of each of the digits after dimensionality reduction and analyze.

Solution (a):

Percentage Variance Retained after dimensionality reduction is **84.76%(**with Standardized Scaling of the training images**)** and **92.84 %(**without Standardized scaling**)**. The plots below depict results with Standardized Scaling.

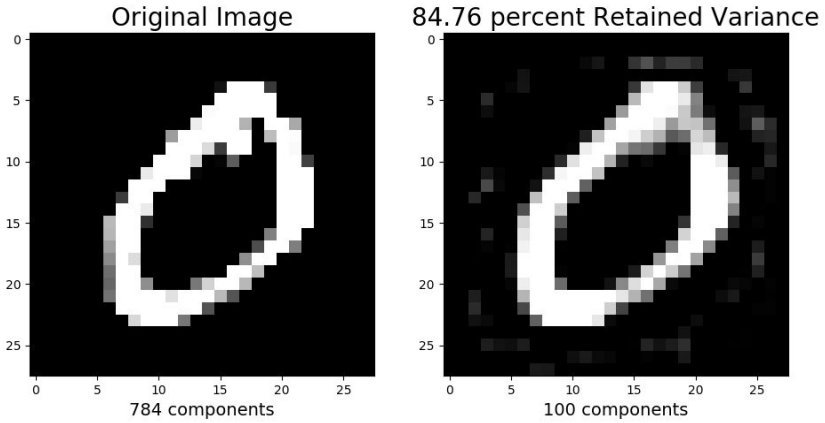
<u>Plot of the covariance matrix for the training data after dimensionality reduction:</u>

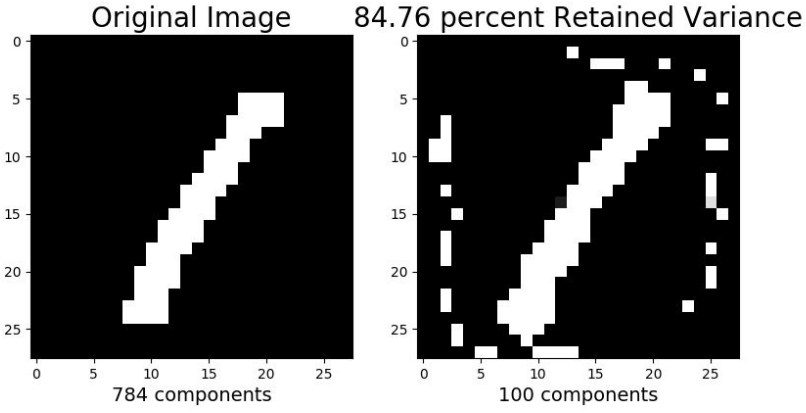


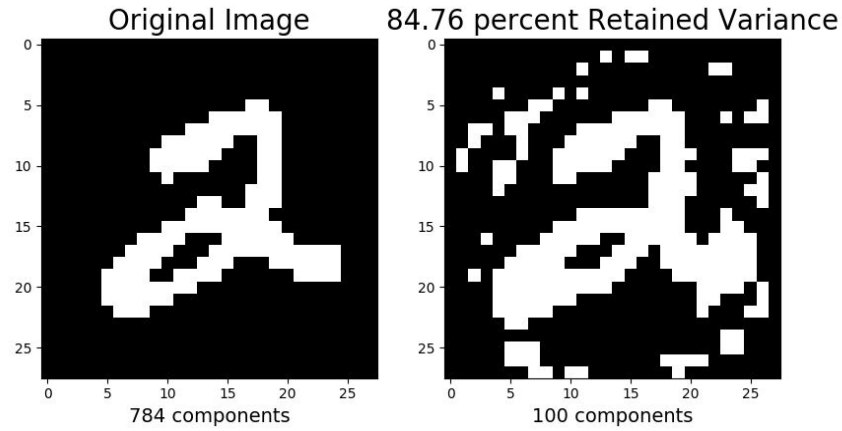
The covariance matrix gives us the relation between points present in a space, and how they change with respect to the change in the other. In this case, the variables are the 1000 training images with reduced dimensions(100), and the matrix tells us how they vary with each other. Here, the covariance matrix is a diagonal matrix, which tells us that that the images do not vary with each other, but only with itself.

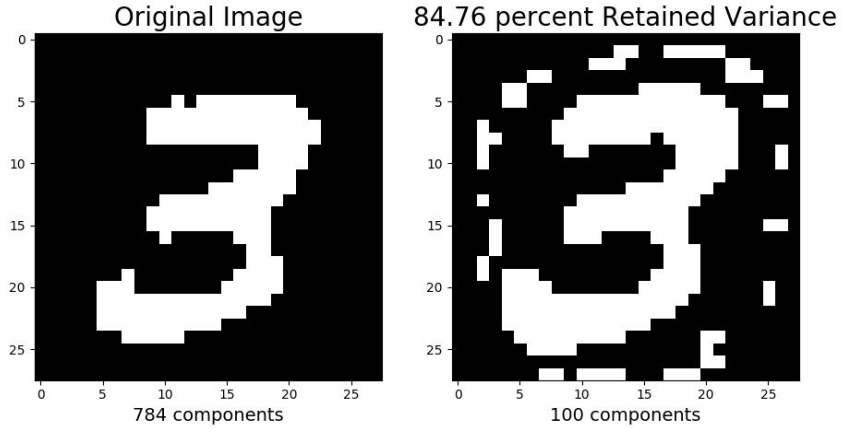
Plots of each digit after dimensionality reduction:

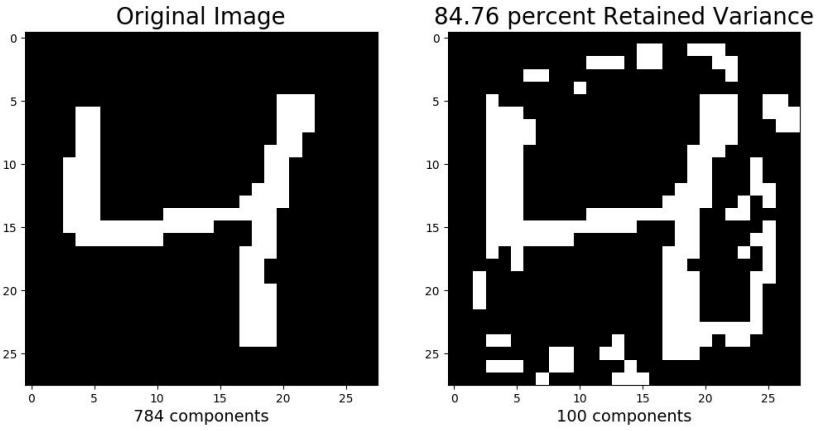
After reducing the principal components from 784 to 100, we observe that 84.76 % of the variance is retained. In the reduced dimensional space, we observe that there is loss because of the removal of principal components which correspond to dimensions with low variation.

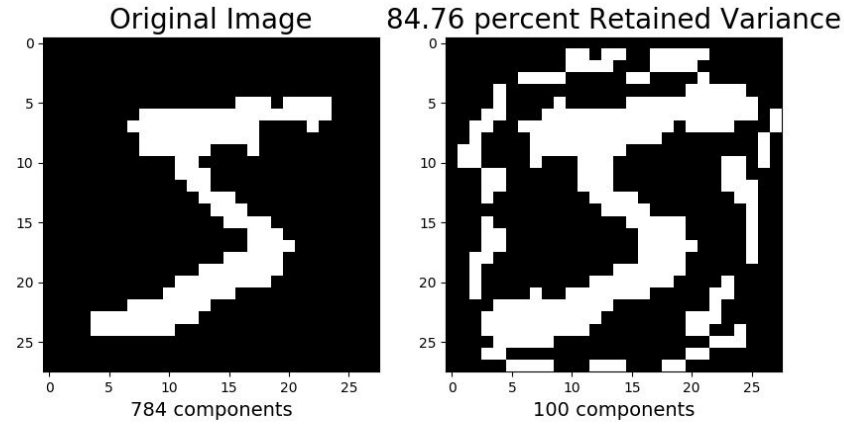


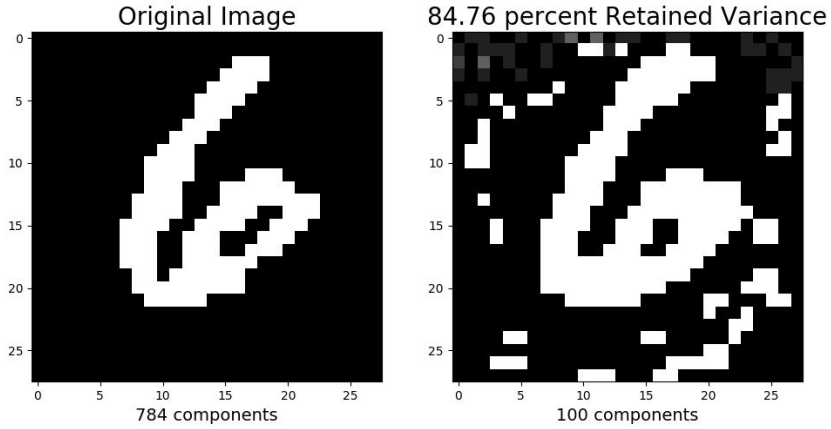


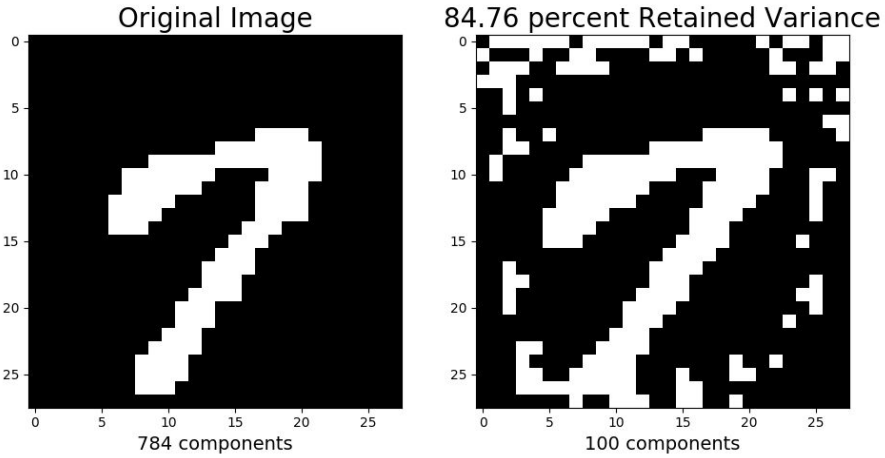

Original Image
84.76 percent Retained Variance
784 components
100 components

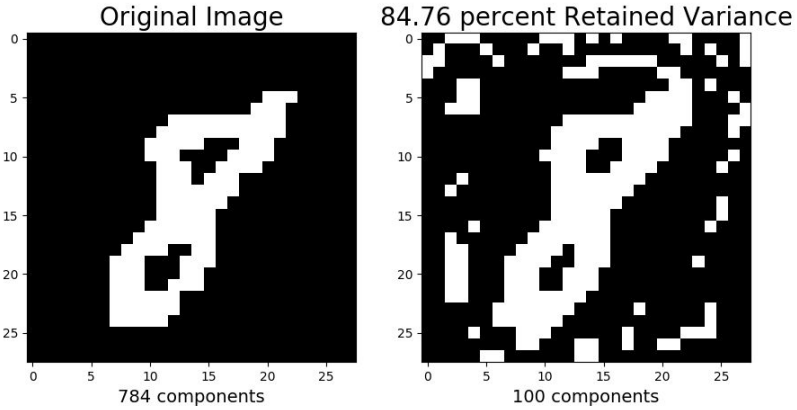

Original Image
84.76 percent Retained Variance
784 components
100 components

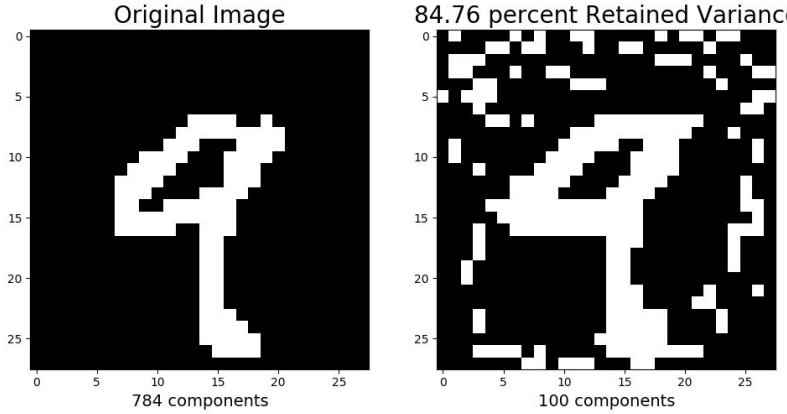

Original Image
84.76 percent Retained Variance
784 components
100 components


Original Image
84.76 percent Retained Variance
784 components
100 components


Original Image
84.76 percent Retained Variance
784 components
100 components


Original Image
84.76 percent Retained Variance
784 components
100 components

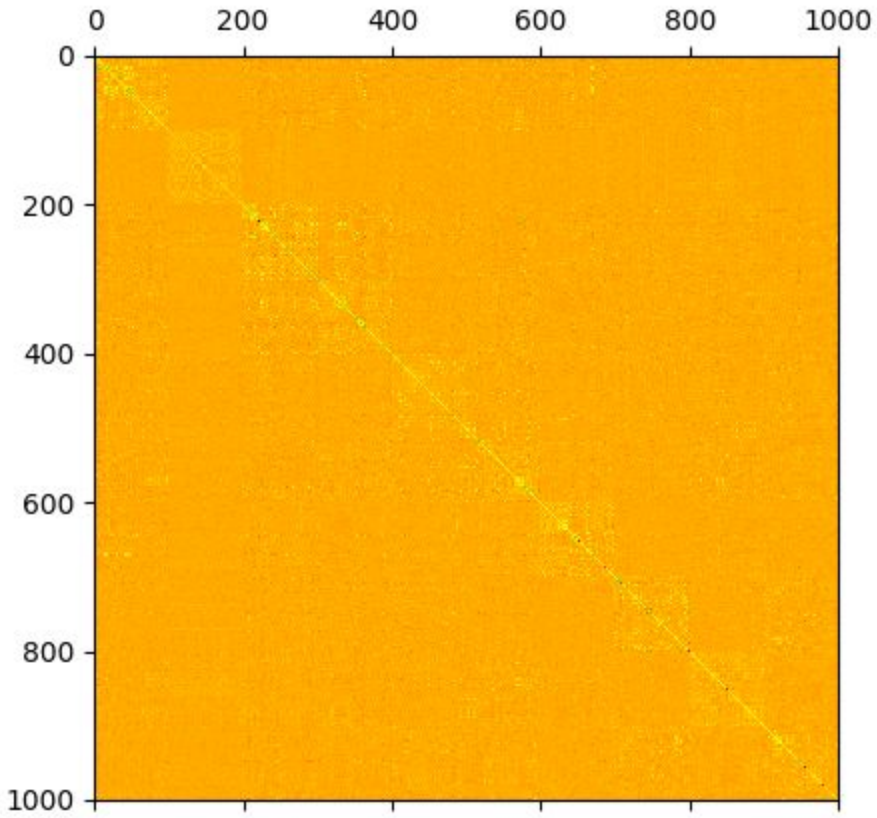

Original Image
84.76 percent Retained Variance
784 components
100 components


Original Image
84.76 percent Retained Variance
784 components
100 components

b) Perform PCA whitening on the reduced dimensions. Plot the covariance matrix after whitening and analyze. Plot one example of each of the digits after PCA whitening and analyze. Implement ZCA whitening and plot the covariance matrix. Plot one example of each of the digits after ZCA whitening and analyze.

Solution (b):

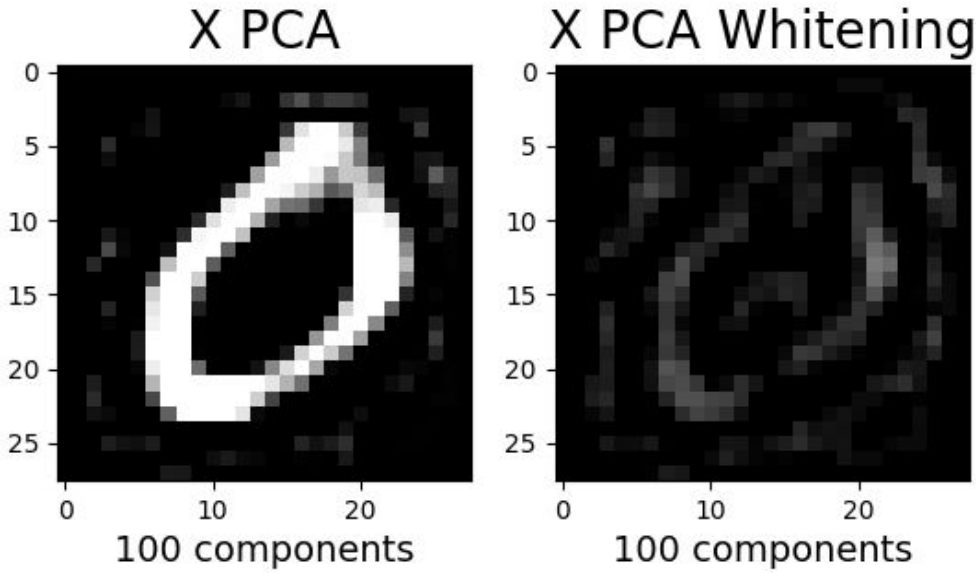
Covariance Matrix Plot after performing PCA Whitening:

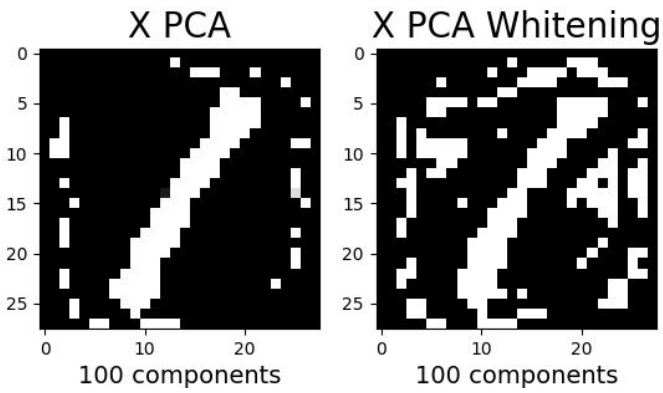


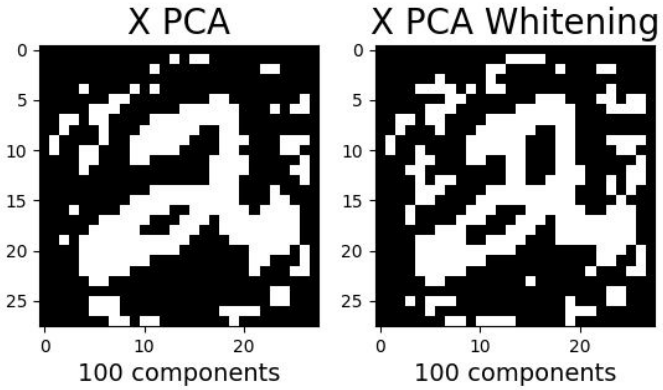
Similar to the covariance after using PCA alone, PCA with whitening also results in a diagonal matrix, which explains the same reason of variation in data with each other. Here, we are whitening with regularization to avoid numerical instability. Due to this, some diagonal values will be lesser than 1, which would have been 1 for all diagonal values in case of $PCA_{Whitening}$ without regularization. In the plot above, we observe a bright yellow diagonal, with some points of different colors and intensities. These points illustrate the diagonal values lesser than 1 due to regularization.

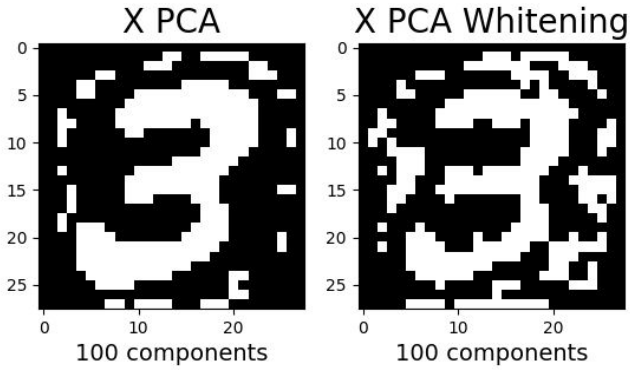
Plots of each digit after PCA Whitening:

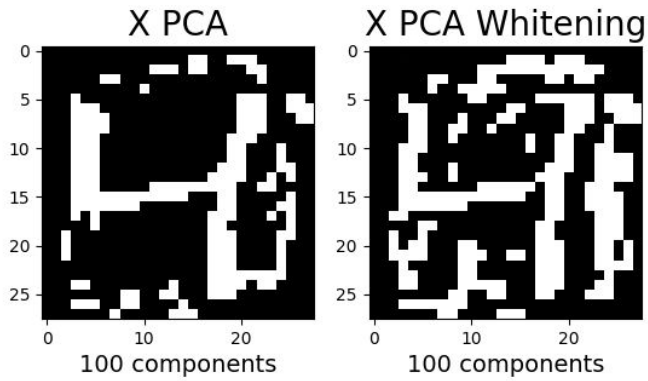
With the aid of PCA Whitening, we aim to remove redundancies of the raw data, in which the adjacent pixels are highly correlated. Thus, it makes the input less redundant. The following plots illustrate the effect of PCA whitening with regularization.

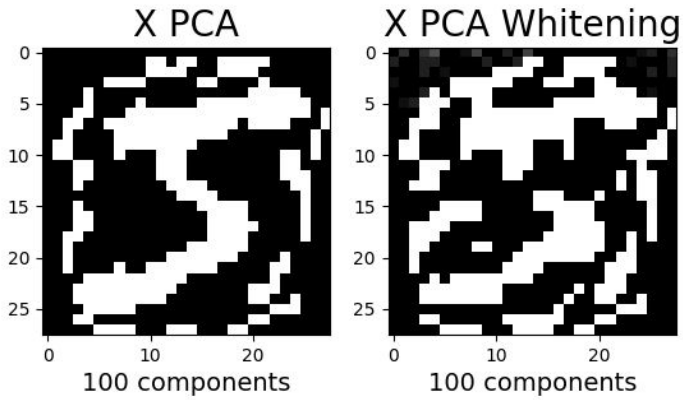


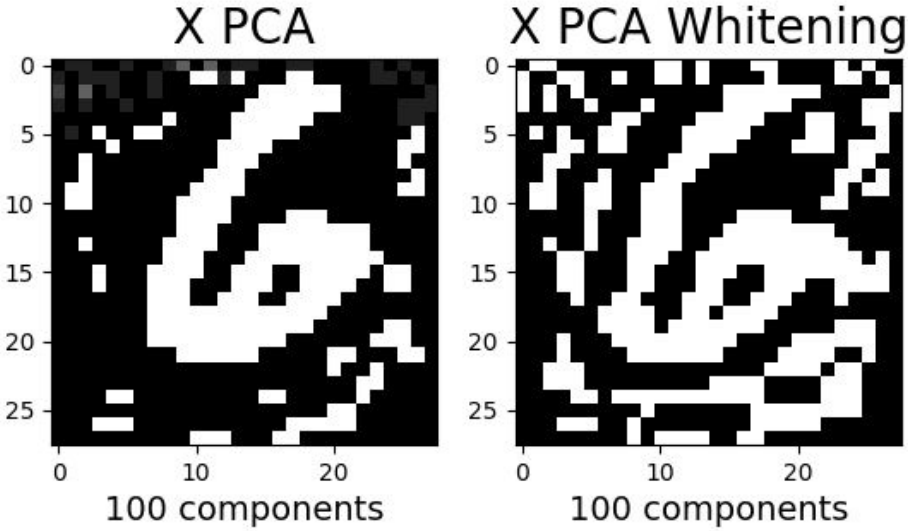

X PCA
X PCA Whitening
100 components
100 components

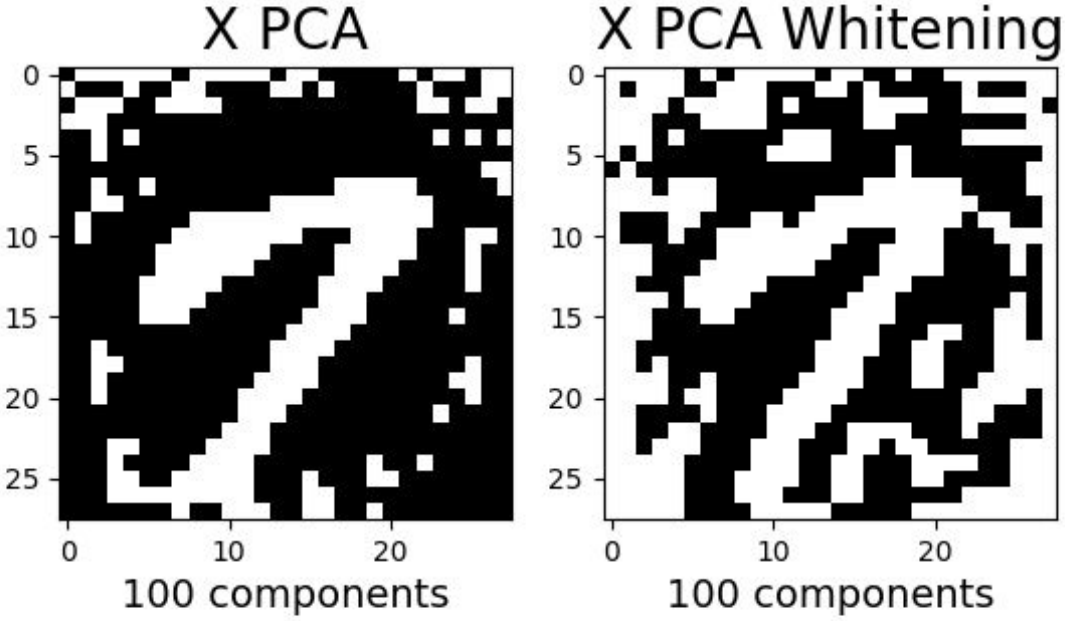

X PCA
X PCA Whitening
100 components
100 components

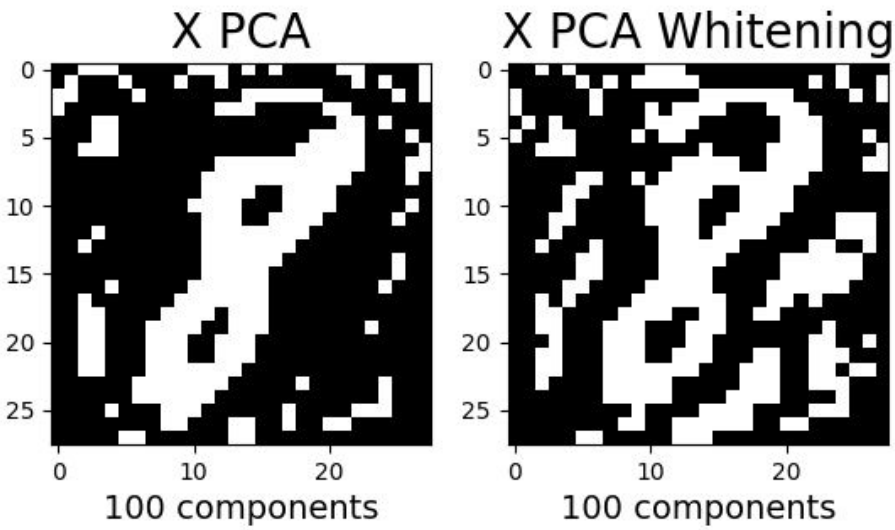

X PCA
X PCA Whitening
100 components
100 components

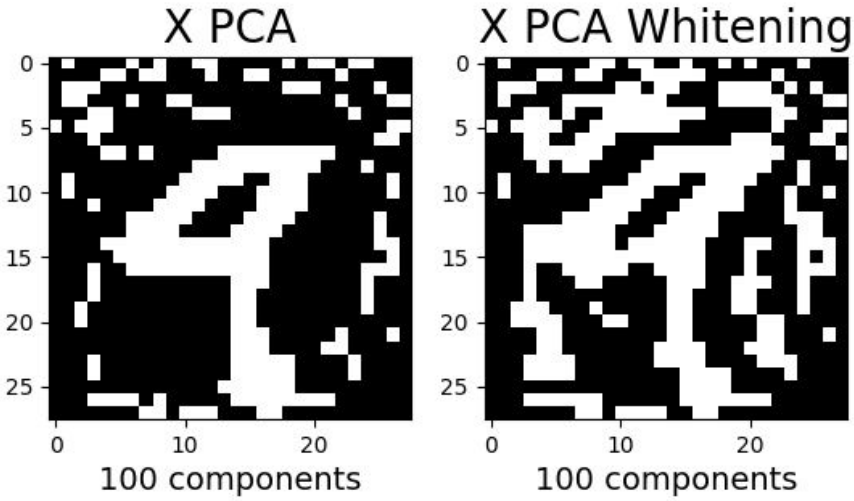

X PCA
X PCA Whitening
0
5
10
15
20
25
0
10
20
100 components
100 components


X PCA
X PCA Whitening
0
5
10
15
20
25
0
10
20
100 components
100 components


X PCA
X PCA Whitening
0
5
10
15
20
25
0
10
20
100 components
100 components

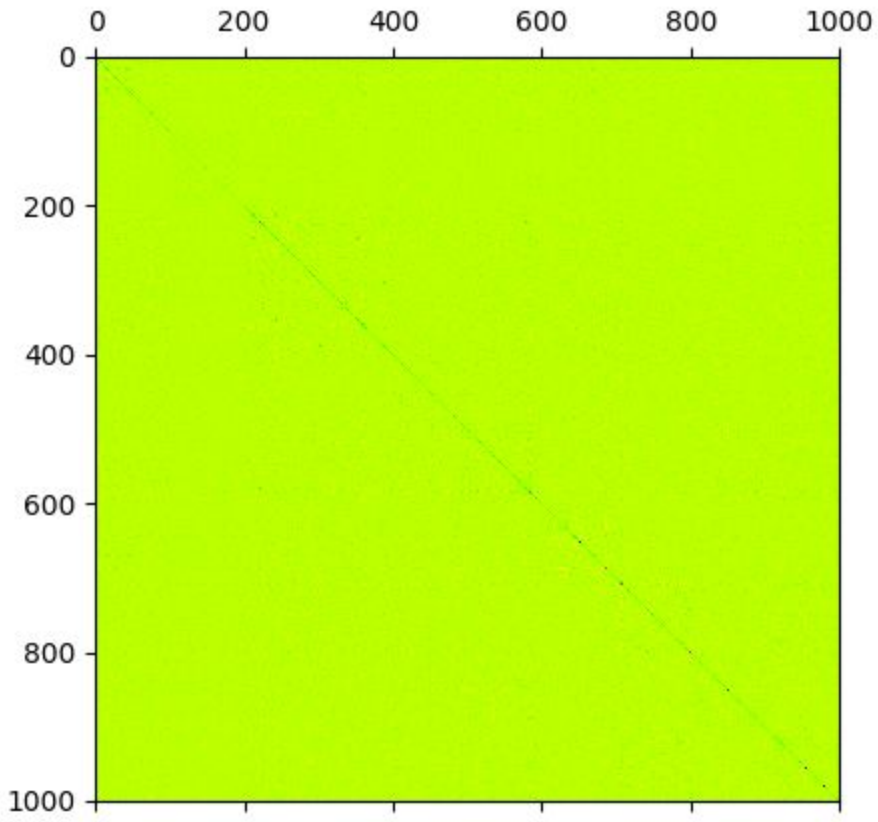

X PCA
X PCA Whitening
0
5
10
15
20
25
0
10
20
100 components
100 components


X PCA
X PCA Whitening
0
5
10
15
20
25
0
10
20
100 components
100 components


X PCA
X PCA Whitening
0
5
10
15
20
25
0
10
20
100 components
100 components

ZCA Whitening:

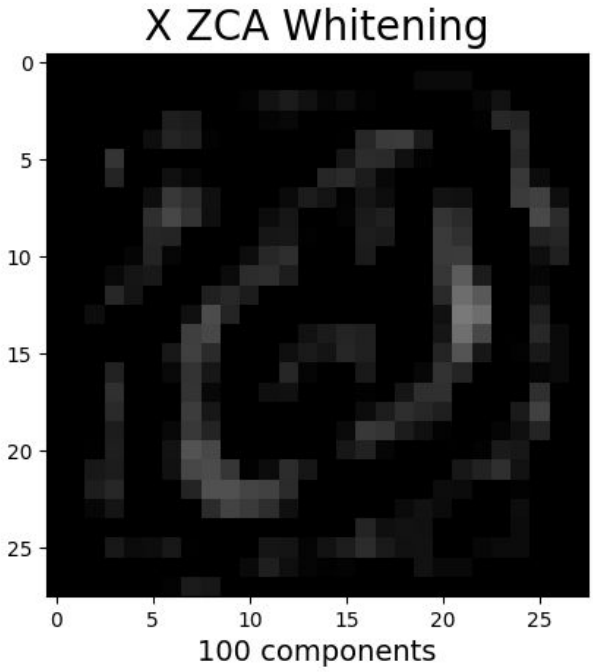
- **Covariance Matrix Plot:**

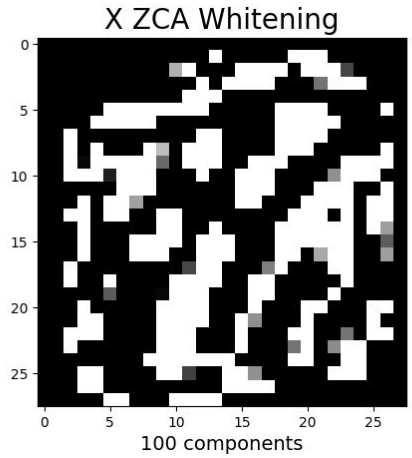


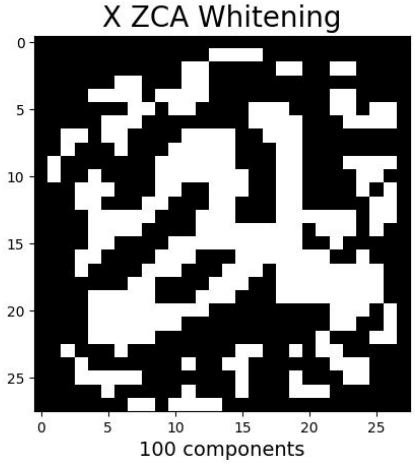
We again get a similar diagonal matrix, which illustrates the lack of variation in data with respect to each other other than itself. With ZCA whitening, we attempt to project the PCA-whitened data back in the original space.

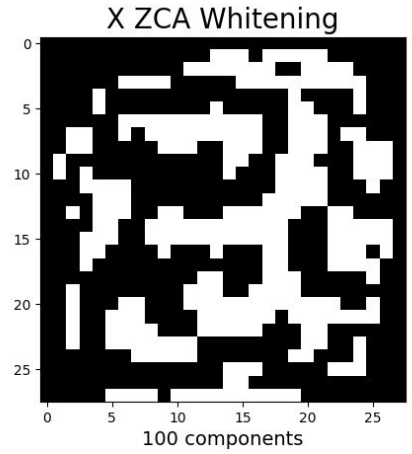
- **Plots of each digit after ZCA Whitening:**

  With ZCA Whitening, we attempt to project the reduced-dimensional PCA-whitened data back in the original space. It is a rotation. We achieve it by taking the product of the $X_{PCAWhitened}$ with the eigen vector E. If we compare the ZCA-whitened results with the original data, we observe more enhanced edges in the results with ZCA-whitening.

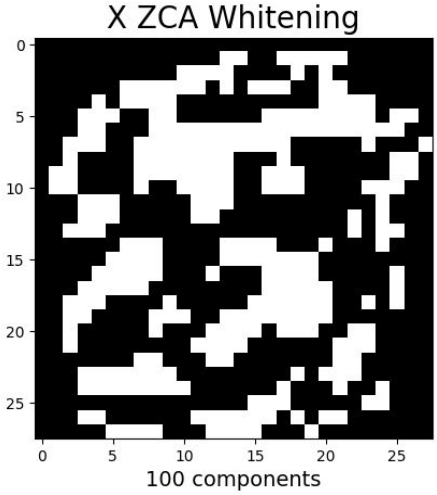


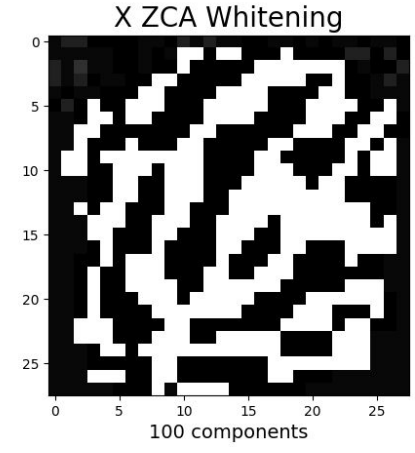

X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components

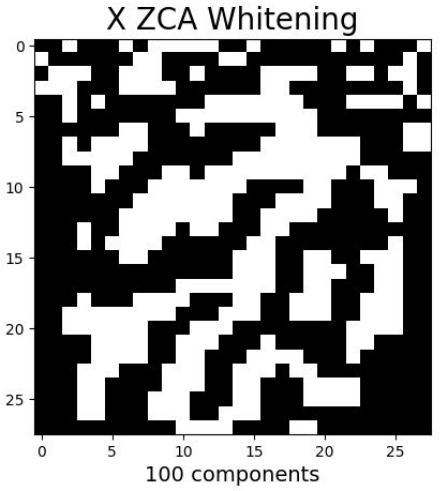

X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components

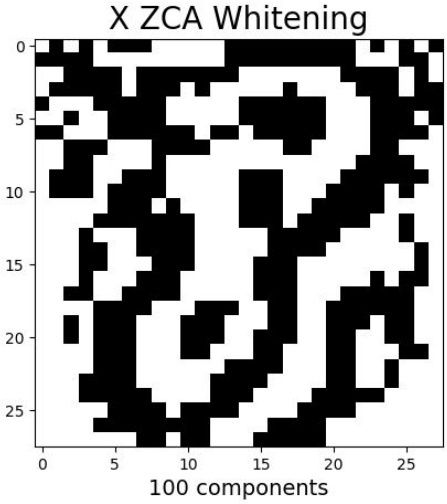

X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components

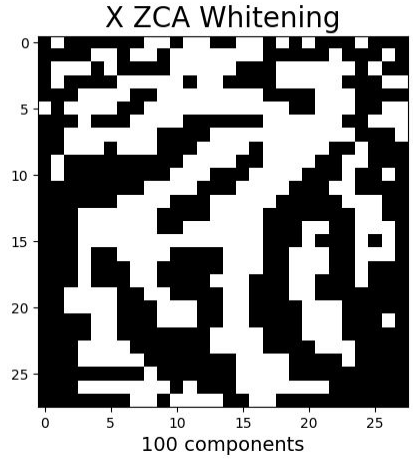

X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components


X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components


X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components


X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components


X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components


X ZCA Whitening
0
5
10
15
20
25
0
5
10
15
20
25
100 components

- **Code Snippet for Part (a) and (b):**

```
import numpy as np
import os
import pdb
import matplotlib.pyplot as plt
import cv2
import math
from load_mnist_updated_4Oct2018 import mnist
from load_mnist_updated_4Oct2018 import one_hot
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
datasets_dir = '/home/kunal/Desktop/CSE569_HW2/data/'



def calculate_percentVarianceRetained(pca1,pca2):
      return (sum(pca1.explained_variance_)/sum(pca2.explained_variance_))*100;

def main():

      #generating mentioned training & testing samples
    trX, trY, tsX, tsY = mnist(noTrSamples=1000,
                               noTsSamples=100, digit_range=[0, 10],
                               noTrPerClass=100, noTsPerClass=10)

    #taking transpose to fit the required matrix format
    trX=trX.T
    tsX=tsX.T

    #pca object to reduce image to 100 components
    pca=PCA(n_components=100)

    #pca object having 784 components
    pca2=PCA(n_components=784)

    #scaling the input to fit the transform
    scalar= StandardScaler()
    trX=scalar.fit_transform(trX)

    #fitting data into pca of 100 components
    eTx=pca.fit_transform(trX)

    #Transforming reduced data back to its original space
    #for plotting and comparison with original
    apX=pca.inverse_transform(eTx)
```

```
    #fitting data into pca of 784 components
    ld2=pca2.fit_transform(trX)

    #Transforming reduced data back to its original space
    #for plotting and comparison with original
    apX2=pca2.inverse_transform(ld2)

    #calculating retained % variance
    percVretained=calculate_percentVarianceRetained(pca,pca2);
    percVretained=round(percVretained,2)
    print percVretained

    #printing the covariance matrix
    cov=getCovariance(eTx)


    #Performing PCA Whitening
    Xpca_w=perform_PCA_Whitening(pca,eTx,cov)

    #Transforming reduced data back to its original space
    #for plotting and comparison with original
    apX3=pca.inverse_transform(Xpca_w)
    cov2=getCovariance(Xpca_w)


    #Performing ZCA Whitening
    Xzca_w=perform_ZCA_Whitening(pca,Xpca_w,cov)
    cov3=getCovariance(Xzca_w)
    displayImgs(apX3,Xzca_w,percVretained)

def perform_ZCA_Whitening(pca,Xpca_w,cov):
        eigen_vectors=pca.components_
        return np.dot(Xpca_w,eigen_vectors)

def perform_PCA_Whitening(pca,eTx,cov):
       eigen_vectors=pca.components_
       eigen_values=pca.explained_variance_
       x=np.dot(eTx,eigen_vectors)
       for i in range(0,100):
              if(eigen_values[i]==0):
                     eigen_vectors[i]=eigen_vectors[i]/math.sqrt(eigen_values[i]+1e-3);
              else:
                     eigen_vectors[i]=eigen_vectors[i]/math.sqrt(eigen_values[i]);

       return np.dot(x,eigen_vectors.T)
```

```
def getCovariance(apX):
      covarianceMat=np.cov(apX)
      plt.matshow(covarianceMat*255,cmap=plt.cm.hsv) # hsv, RdBu
      plt.show()

def displayImgs(trX,apX,percVretained):
      i=0
      while i < 999:
            # image1
          plt.subplot(1, 2, 1);
          trX=trX*255;
          plt.imshow(trX[i].reshape(28,28),cmap = plt.cm.gray,
            interpolation='nearest',
                        clim=(0, 255));
          plt.xlabel('100 components', fontsize = 14)
          plt.title('X PCA Whitening', fontsize = 20);

            # image2
          apX=apX*255;
          plt.subplot(1, 2, 2);
          plt.imshow(apX[i].reshape(28, 28),cmap = plt.cm.gray,
            interpolation='nearest',
                        clim=(0, 255));
          plt.xlabel('100 components', fontsize = 14)
          plt.title('X ZCA Whitening', fontsize = 20);
          plt.show()
          i+=100

if __name__ == "__main__":
    main()
```

c) Implement Fisher's Linear discriminant for digits 0 and 1 in the training dataset. Use reduced PCA dimensions (100). Report the accuracy on the test dataset having only digits 0 and 1.

Solution (c):

Fitting Fisher's Linear Discriminant with 200 images of 0's and 1's in total, and testing them on 20 images of 0s and 1s from a testing set of 100 images, we get an accuracy of **100 percent** on the test dataset having only digits 0 and 1.

Code Snippet for this Problem: On the next Page

```
import numpy as np
import os
import pdb
import matplotlib.pyplot as plt
import cv2
import math
from load_mnist_updated_4Oct2018 import mnist
from load_mnist_updated_4Oct2018 import one_hot
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import matplotlib as mpl
from matplotlib import colors

datasets_dir = '/home/kunal/Desktop/CSE569_HW2/data/'


def main():

      #generating mentioned training & testing samples
    trX, trY, tsX, tsY = mnist(noTrSamples=1000,
                               noTsSamples=100, digit_range=[0, 10],
                               noTrPerClass=100, noTsPerClass=10)

    #taking transpose to fit the required matrix format
    trX=trX.T
    tsX=tsX.T
    tsY=tsY.T

    #generating training images and labels of only 0's and 1's
    trX_01=trX[0:200]
    trY_01=trY[0][0:200]
    trX_01=np.array(trX_01)
    trY_01=np.array(trY_01)

    # Empty list for finding out test images and labels of only 0s
    # and 1s
    tsX_01=[]
    tsY_01=[]

    #pca object to reduce image to 100 components
    pca=PCA(n_components=100)

    #scaling the input to fit the transform
    scalar= StandardScaler()
    trX_01=scalar.fit_transform(trX_01)
    tsX=scalar.fit_transform(tsX)
```

```
    #fitting data into pca of 100 components
    eTx=pca.fit_transform(trX_01)

    # Fitting training images to Fisher's
    # Linear Discriminant space
    fLD= LinearDiscriminantAnalysis()
    fLD.fit(eTx,trY_01)

    #reducing test set to 100 components
    reduced_tsX=pca.fit_transform(tsX)


    # collecting only the 0's and 1's from the
    # test images
    for i in range(0,100):
       if tsY[i]==0 or tsY[i]==1:
              tsX_01.append(reduced_tsX[i])
              tsY_01.append(tsY[i])

    #Converting the list object to an nparray
    tsX_01=np.array(tsX_01)
    tsY_01=np.array(tsY_01)

    #Predicting results on the test-set
    results=fLD.predict(tsX_01)

    #Results
    print results
    print fLD.score(tsX_01,tsY_01)*100

if __name__ == "__main__":
    main()
```
