REPORT

Vrushali Pandit - B20BB047

Lab 7 - Principle Component Analysis

Data:

1. MNIST dataset of handwritten images 28x28 pixels
2. Shape = 60,000 images with 28*28 = 784 features each

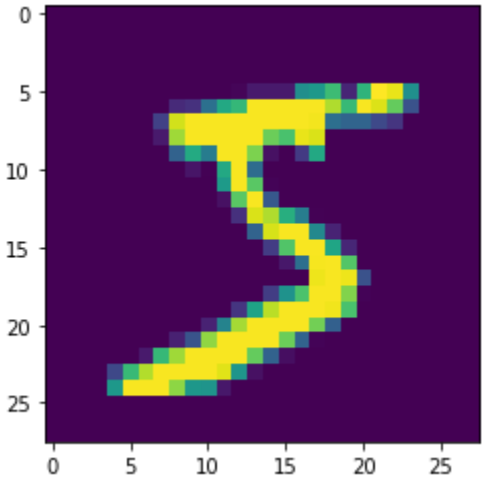


Figure 1

Task 1:

Data Visualization :

1. Downloaded the dataset from sklearn library using the function load_digits()
2. This gives us a sklearn utils package which involves the data, its ground truth labels and images.
3. The images are black and white MNIST like handwritten images, but 8x8 pixels instead of 28x28 pixels. The number of images is also lower.
4. Hence the dataset is 1797 samples and 64 features
5. To visualize the same, we must use numPy's inbuilt function np.reshape(<desired shape>)
6. One such visualization is shown below.

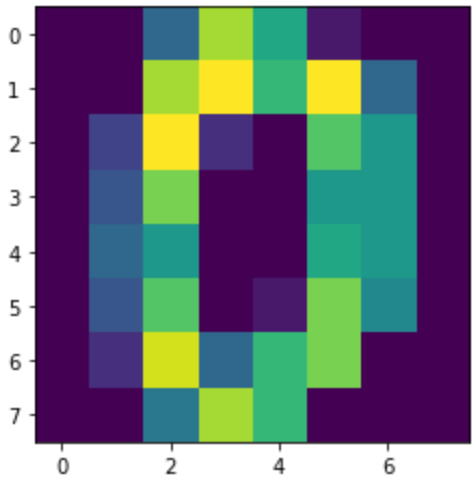


Figure 2

Pre-processing :

1. For PCA, we must apply standard scaling as this shifts the origin to the mean of the data, and ensures reduction in bias.
2. We import StandardScaler() from sklearn.preprocessing to do the same

Applying PCA :

1. We apply PCA using two methods:
    a. Pre-defining the number of components
        i. **array([0.12033916, 0.09561054, 0.08444415, 0.06498408, 0.04860142])**
        ii. The array of variance explained by the first 5 components
    b. By defining how much variance we wish to retain

Task2:

Data Visualization :

1. Data is down-loaded from https://www.kaggle.com/oddrationale/mnist-in-csv
2. This has 60,000 handwritten images of 28x28 pixels, with 785 features, one being the target label feature.
3. Similarly as in Task1 data visualization part, we have to reshape the data to visualize it using np.reshape(28,28)
4. One such visualization is shown in **Figure 1**

Pre-processing:

1. Take a sample of 5000 images
2. Apply standard Scaling on the dataset. This is to ensure that the origin is shifted to the mean sample and all features are equally considered with little bias.
3. After scaling, we find the covariance matrix of it. I have done this from scratch using np.mean() and np.std()
4. Z = the matrix obtained in step 3. Z is a 784x784 sized matrix.
5. Eigendecomposition of Z is done using a numPy function np.linalg.eig(Z)
6. The eigenvectors and eigenvalues obtained are all complex numbers (all with imaginary parts = 0)
7. We extract just the real parts for the same using np.real()

PCA:

1. We arrange the Eigenvectors in descending order of their eigenvalues using a simpleSort
2. Then we sample any two random images from the standard Scaled original data.

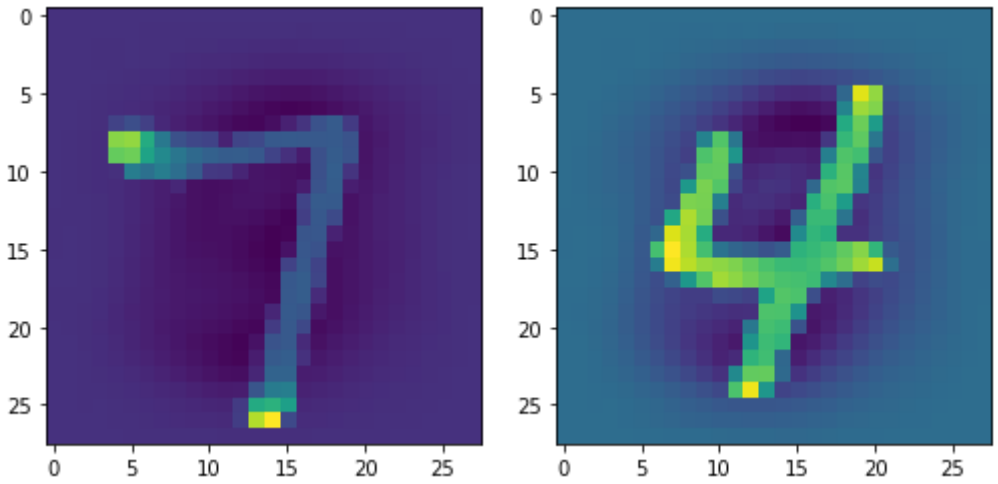
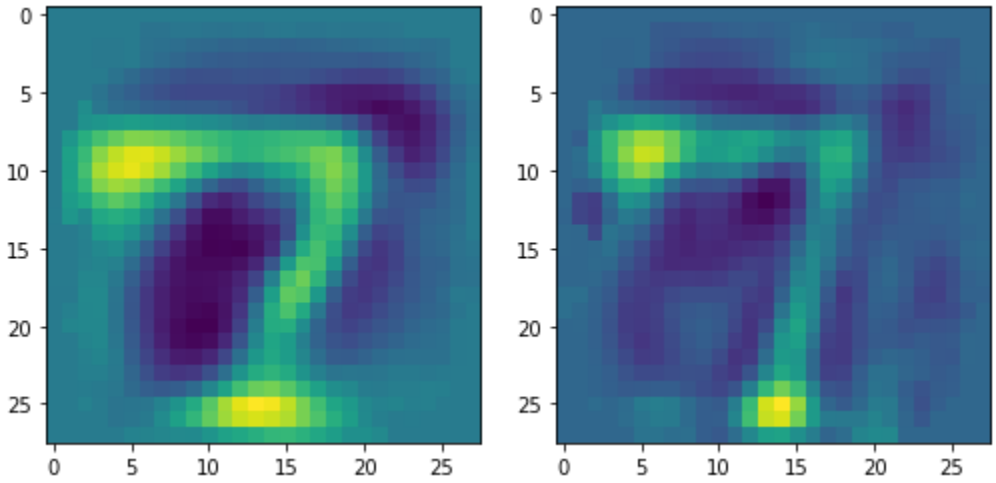


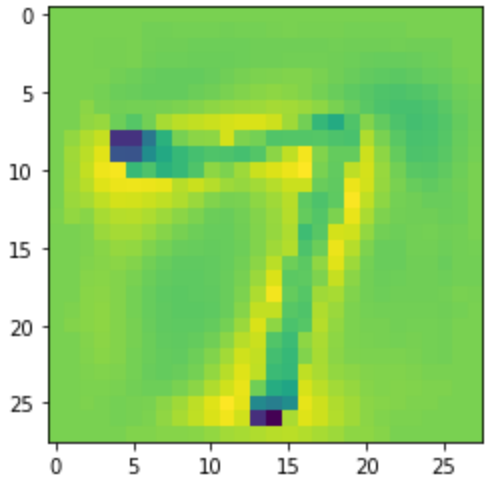
Figure 3

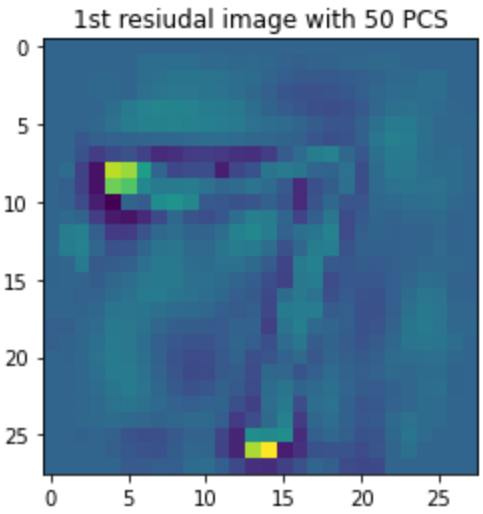
3. From the chosen number of eigen vector components or Principle components to be retained, our new features will be the dot product of the original sample and the chosen eigen components.
4. Hence we now have the same number of samples but with features = number of PC's retained. The code for this is written as **reduce_()**
5. Applying reduce_() for different values of the number of components to be retrained and for the above two images we get new reduced_imgi_j which are in the lower dimensional space.
6. We must scale these lower dimensional space images to the higher dimension by using Inverse PCA. Here we take the dot product of the reduced_img with the the eigenvector matrix. This does the inverse transform of step 5

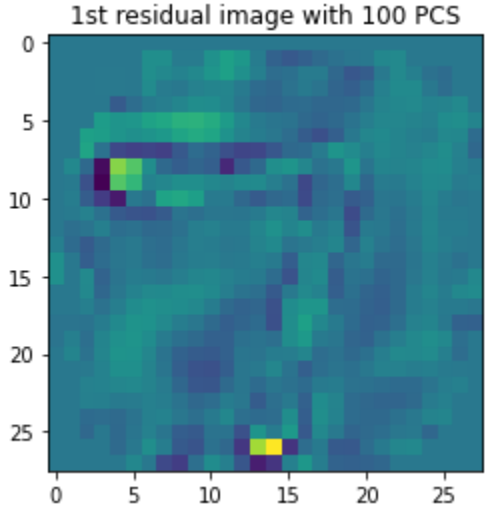
7. We get such images by the same
   a. 

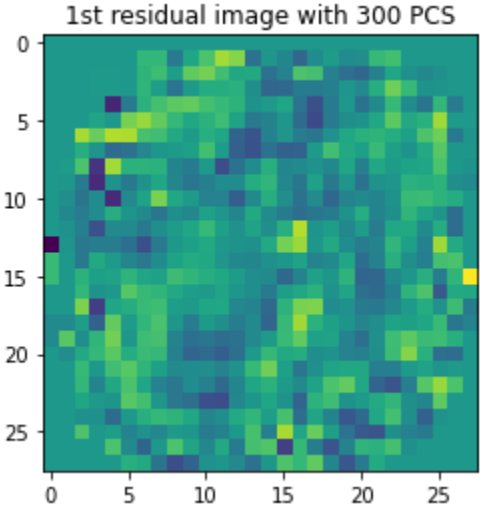
10 components retained 50 components retain

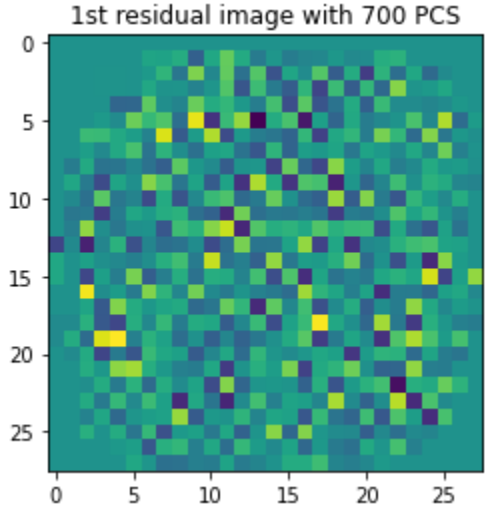
8. These are the reconstructed images.
9. Now to see the kind of error or loss of information PCA brings about we see the residual images.
   a. RESIDUAL IMAGES:

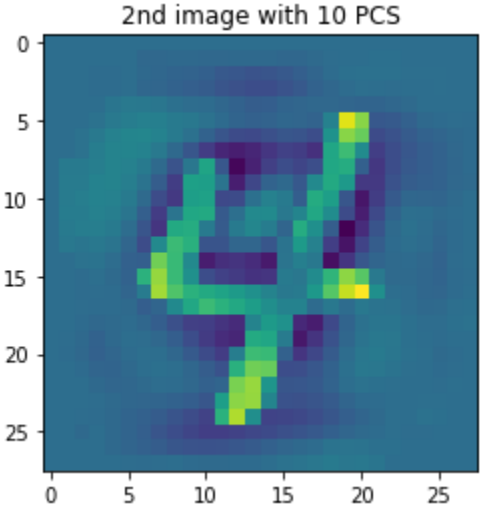


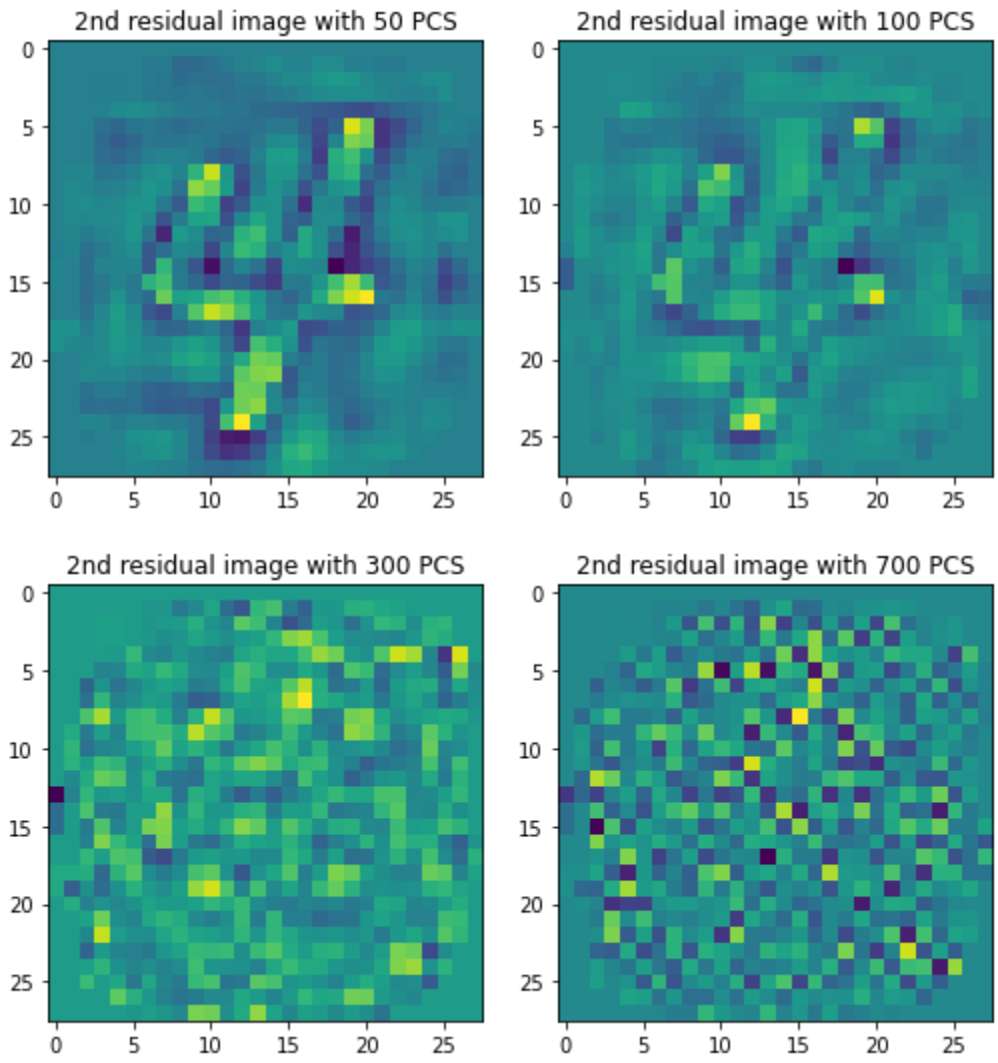

1st resiudal image with 50 PCS


1st residual image with 100 PCS

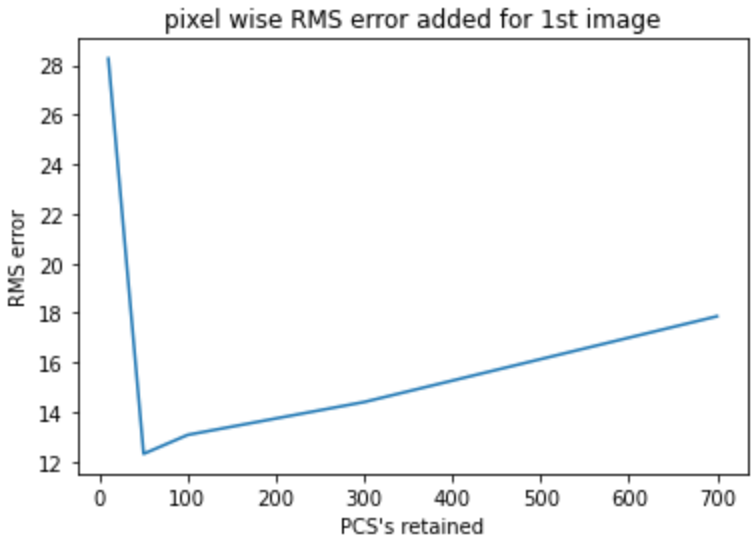

1st residual image with 300 PCS

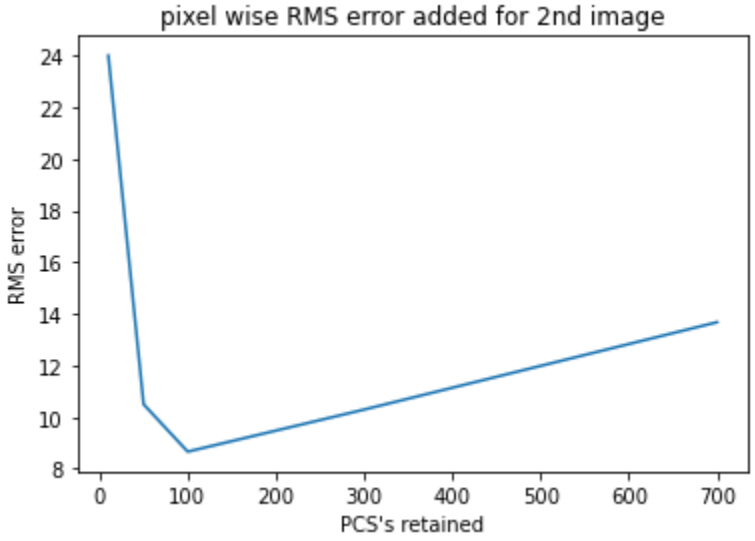

1st residual image with 700 PCS


2nd image with 10 PCS



10. Now we find the root mean square error for the residual images and sum up for each pixel and plot against number of components taken





Explanation of Results:

- With too few components retained we capture too little variance and hence this gives a lot of error in reconstruction.
- Then the error takes a minimum where we get the best number of components to approximate the images with
- Later the error starts increasing. This could be because we are including more features than we require to hence adding to noise.