```
!unzip archive.zip -d archive
```

```
Archive:  archive.zip
  inflating: archive/README
  inflating: archive/s1/1.pgm
  inflating: archive/s1/10.pgm
  inflating: archive/s1/2.pgm
  inflating: archive/s1/3.pgm
  inflating: archive/s1/4.pgm
  inflating: archive/s1/5.pgm
  inflating: archive/s1/6.pgm
  inflating: archive/s1/7.pgm
  inflating: archive/s1/8.pgm
  inflating: archive/s1/9.pgm
  inflating: archive/s10/1.pgm
  inflating: archive/s10/10.pgm
  inflating: archive/s10/2.pgm
  inflating: archive/s10/3.pgm
  inflating: archive/s10/4.pgm
  inflating: archive/s10/5.pgm
  inflating: archive/s10/6.pgm
  inflating: archive/s10/7.pgm
  inflating: archive/s10/8.pgm
  inflating: archive/s10/9.pgm
  inflating: archive/s11/1.pgm
  inflating: archive/s11/10.pgm
  inflating: archive/s11/2.pgm
  inflating: archive/s11/3.pgm
  inflating: archive/s11/4.pgm
  inflating: archive/s11/5.pgm
  inflating: archive/s11/6.pgm
  inflating: archive/s11/7.pgm
  inflating: archive/s11/8.pgm
  inflating: archive/s11/9.pgm
  inflating: archive/s12/1.pgm
  inflating: archive/s12/10.pgm
  inflating: archive/s12/2.pgm
  inflating: archive/s12/3.pgm
  inflating: archive/s12/4.pgm
  inflating: archive/s12/5.pgm
  inflating: archive/s12/6.pgm
  inflating: archive/s12/7.pgm
  inflating: archive/s12/8.pgm
  inflating: archive/s12/9.pgm
  inflating: archive/s13/1.pgm
  inflating: archive/s13/10.pgm
  inflating: archive/s13/2.pgm
  inflating: archive/s13/3.pgm
  inflating: archive/s13/4.pgm
  inflating: archive/s13/5.pgm
  inflating: archive/s13/6.pgm
  inflating: archive/s13/7.pgm
  inflating: archive/s13/8.pgm
  inflating: archive/s13/9.pgm
  inflating: archive/s14/1.pgm
  inflating: archive/s14/10.pgm
  inflating: archive/s14/2.pgm
  inflating: archive/s14/3.pgm
  inflating: archive/s14/4.pgm
  inflating: archive/s14/5.pgm
```

```
!pip3 install numpy
```

```
!pip3 install pillow
!pip3 install scikit-learn
!pip3 install matplotlib

     Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
     Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (9.4.0)
     Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
     Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from
     Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from s
     Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from
     Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages
     Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
     Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (fr
     Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from m
     Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (f
     Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (f
     Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from ma
     Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (fro
     Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from
     Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (fr
     Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages
     Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from pytho
```

```
import numpy as np
import os
from PIL import Image
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# lists for storing the data matrix D and label vector y
D = []
y = []

# 2) Generate the Data Matrix and the Label vector
for subject in range(1, 41):
    # every subject has 10 images, get 10 images per subject
    imageCount = 0

    for image in os.listdir(f'archive/s{subject}'):
        temp = Image.open(f'archive/s{subject}/{image}')
        vector = np.array(temp).flatten()

        y.append(subject)
        D.append(vector)

# convert the dataMatrix and labels to numpy arrays
D = np.array(D)
y = np.array(y)


# 3) Split the data-set into Training and Test sets
num_images = D.shape[0]
rng_idx = np.random.permutation(num_images)
split= 0.5
split_idx = int(num_images * split)
training_data = D[rng_idx[:split_idx]]
testing_data = D[rng_idx[split_idx:]]

training_labels = y[rng_idx[:split_idx]]
testing_labels = y[rng_idx[split_idx:]]
```

```
testing_labels = y[img_idx[split_idx:]]


# 4) Classification using PCA
# Calculate Projection Matrix U
training_mean = np.mean(training_data, axis=0)
training_std = np.std(training_data, axis=0)
training_centered = training_data - training_mean
covariance_matrix = np.cov(training_centered.T)
```

12 minute operation on T4 GPU

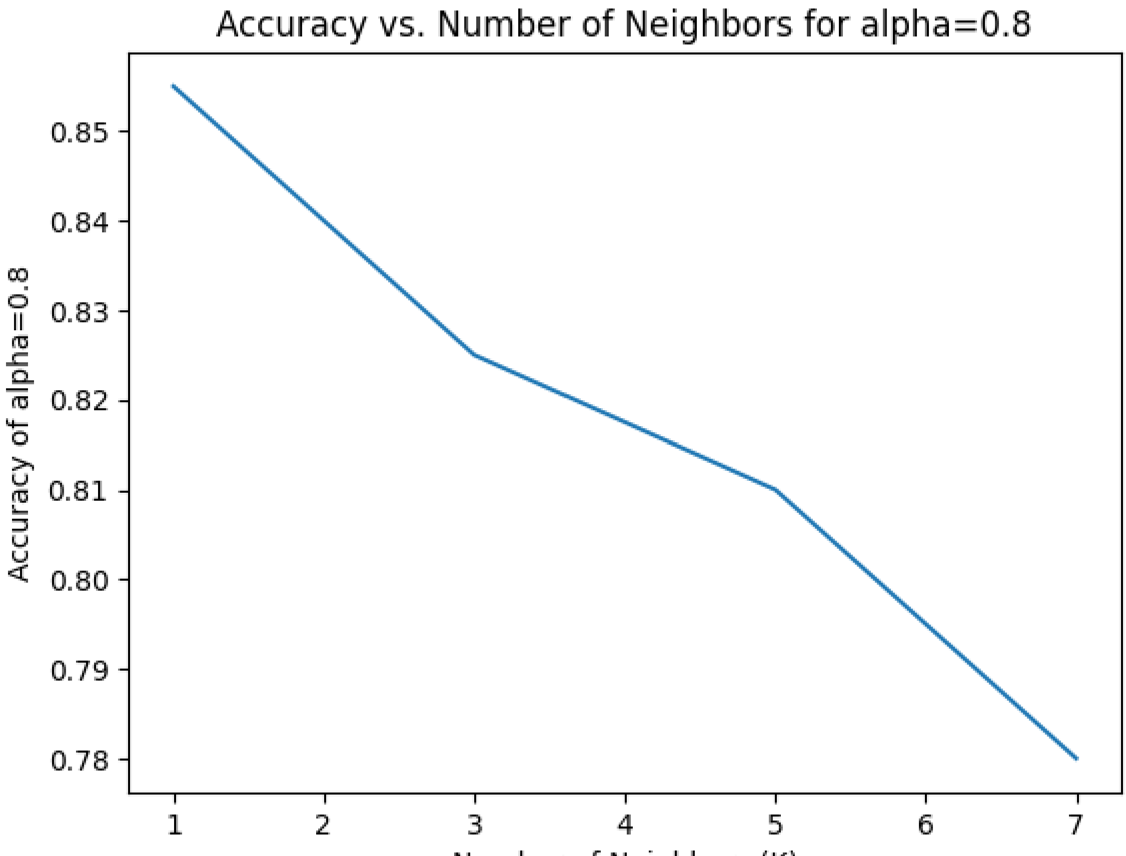
```
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
```

Continue 4)

```
eigenvalues = eigenvalues.real
eigenvectors = eigenvectors.real

# index to sort the eigen values and eigen vectors in decreasing order of eigen values
idx = np.argsort(eigenvalues)[::-1]
sorted_eigenvalues = eigenvalues[idx]
sorted_eigenvectors = eigenvectors[:, idx]

# sum to get the variance fraction to choose how many dimension aka how many eigen vectors
cumulative_sum = np.cumsum(sorted_eigenvalues)


# alpha=[0.8,0.85,0.9,0.95]    loop on the array and mark accuracy
#consider single alpha for now
alphas = [0.8, 0.85, 0.9, 0.95]  # for example

for alpha in alphas:
  # alpha = 0.8
  # Compute the total variance
  total_variance = np.sum(sorted_eigenvalues)

  # Compute the cumulative sum of the sorted eigenvalues
  cumulative_variance = np.cumsum(sorted_eigenvalues)

  # Compute the cumulative proportion of the total variance
  cumulative_proportion = cumulative_variance / total_variance
  print(np.where(cumulative_proportion >= alpha)[0][0])

  num_eigenvectors = np.where(cumulative_proportion >= alpha)[0][0] + 1
  # final eigen vectors chosen for projection

  projected_eigenvectors = sorted_eigenvectors[:,:num_eigenvectors]
  print(len(projected_eigenvectors))
  print(len(eigenvectors))
  D_train_pca = training_centered.dot(projected_eigenvectors)
  #TODO: remember to move testing mean and centered
  testing_mean = np.mean(testing_data, axis=0)
```

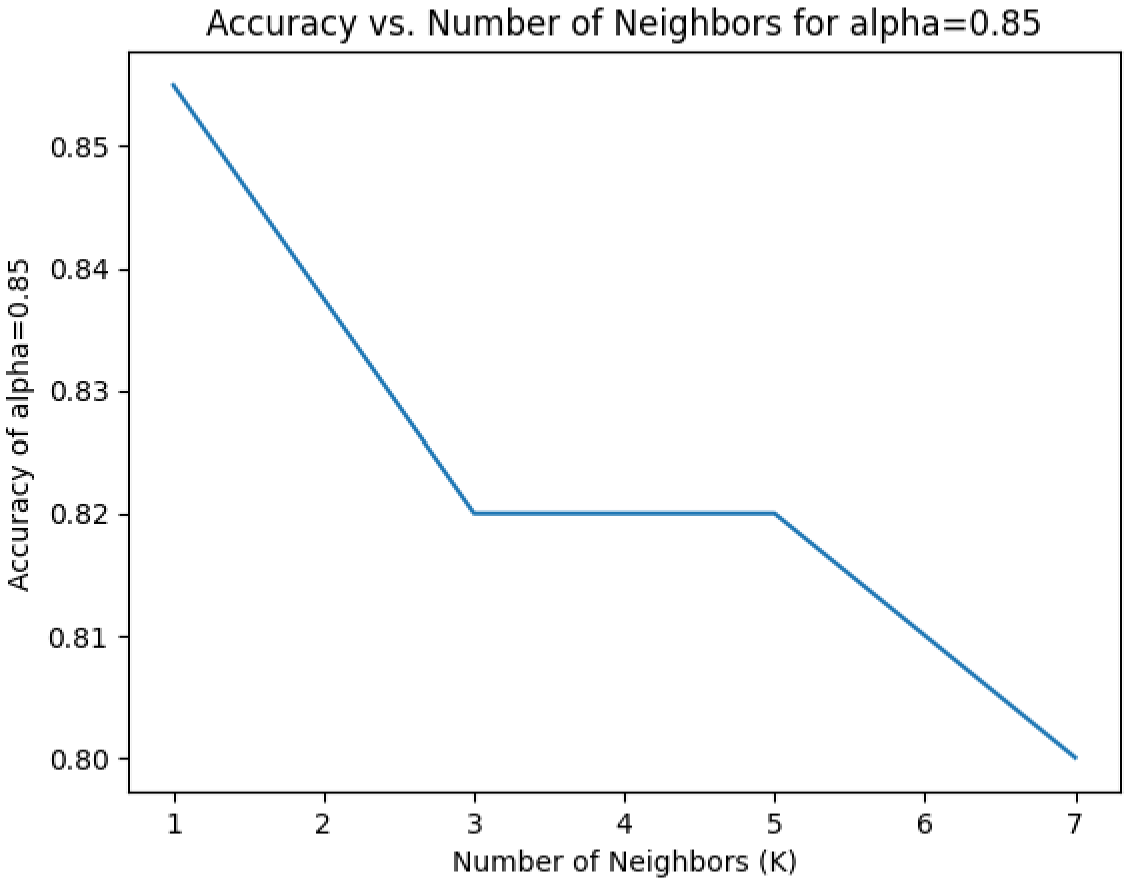
```
testing_centered = testing_data - testing_mean
D_test_pca = testing_centered.dot(projected_eigenvectors)
# U = sorted_eigenvectors[:, :num_eigenvectors]

# project all the data on the eigen vectors
# D_train_pca = np.dot(training_data, U)
# D_test_pca = np.dot(testing_data, U)

# training: fitting the points on the graph so the classifier can classify any new testing point
# 5) Classifier Tuning
knn_nums = [1, 3, 5, 7]
accuracies = []
for knn_num in knn_nums:
    knn = KNeighborsClassifier(n_neighbors=knn_num, weights='distance')
    knn.fit(D_train_pca, training_labels)

    # testing
    predicted_labels = knn.predict(D_test_pca)

    # accuracy
    accuracy = accuracy_score(testing_labels, predicted_labels)
    accuracies.append(accuracy)
    print(f'Accuracy of alpha={alpha}, K={knn_num}: {accuracy}')

plt.plot(knn_nums, accuracies)
plt.xlabel('Number of Neighbors (K)')
plt.ylabel(f'Accuracy of alpha={alpha}')
plt.title(f'Accuracy vs. Number of Neighbors for alpha={alpha}')
plt.show()
```

```
34
10304
10304
Accuracy of alpha=0.8, K=1: 0.855
Accuracy of alpha=0.8, K=3: 0.825
Accuracy of alpha=0.8, K=5: 0.81
Accuracy of alpha=0.8, K=7: 0.78
```

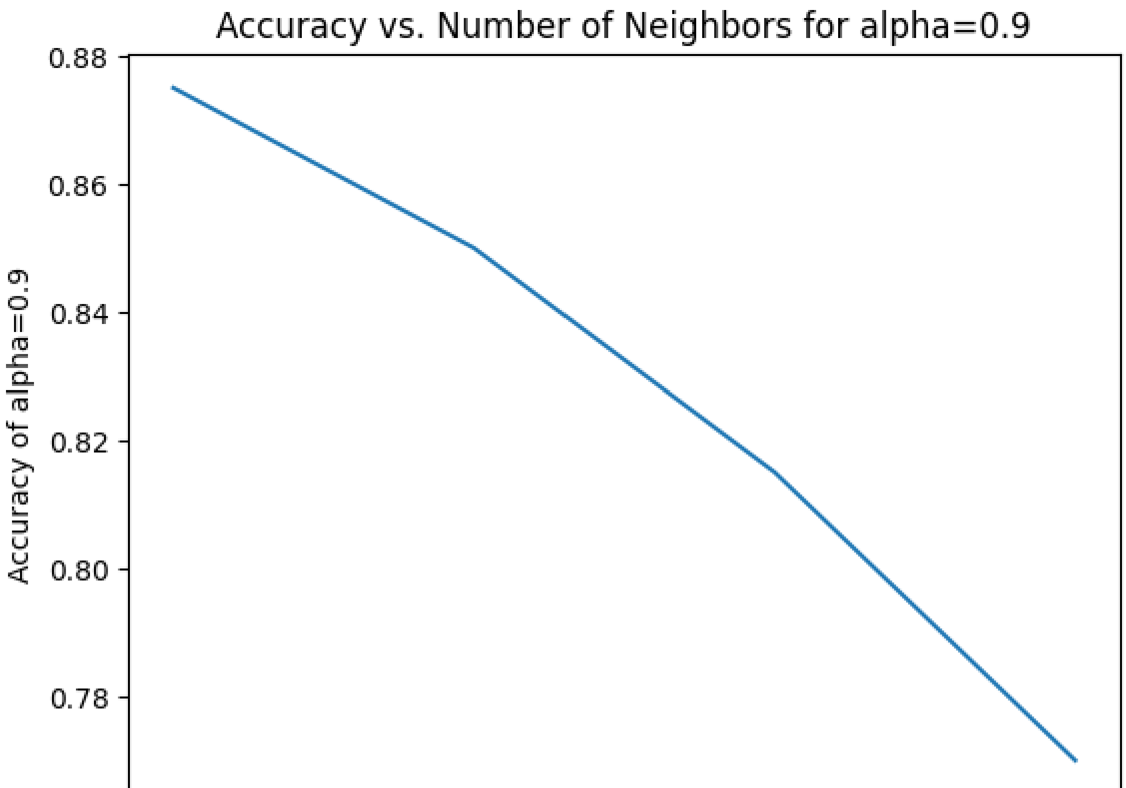


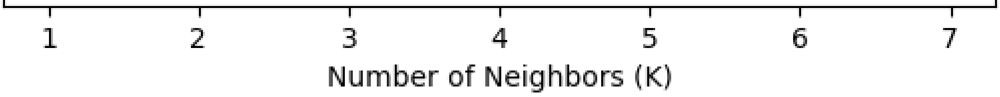
Number of Neighbors (K)

```
50
10304
10304
Accuracy of alpha=0.85, K=1: 0.855
Accuracy of alpha=0.85, K=3: 0.82
Accuracy of alpha=0.85, K=5: 0.82
Accuracy of alpha=0.85, K=7: 0.8
```




```
74
10304
10304
Accuracy of alpha=0.9, K=1: 0.875
Accuracy of alpha=0.9, K=3: 0.85
Accuracy of alpha=0.9, K=5: 0.815
Accuracy of alpha=0.9, K=7: 0.77
```






```
114
10304
10304
Accuracy of alpha=0.95, K=1: 0.86
Accuracy of alpha=0.95, K=3: 0.835
Accuracy of alpha=0.95, K=5: 0.815
Accuracy of alpha=0.95, K=7: 0.76
```