Hashing: Querying in Face Datasets

Name - Amith Korada

Course - AI & ML
(Batch - 4)

Duration - 12 Months

Problem Statement -Implement a basic hashing model from scratch that hashes the images

Prerequisites -

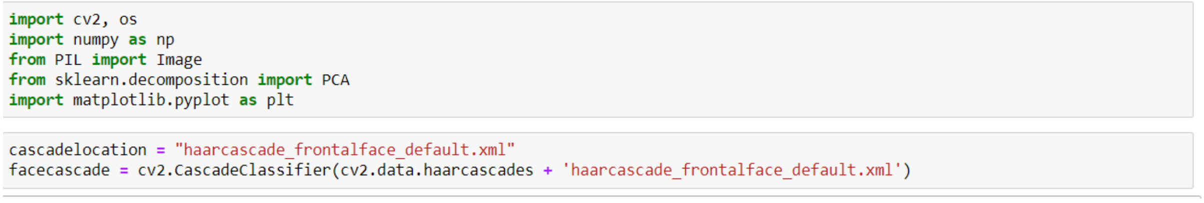
What things you need to install the software and how to install them:

Python 3.6 This setup requires that your machine has the latest version of python. The following URL https://www.python.org/downloads/ can be referred to as download python.

The second and easier option is to download anaconda and use its anaconda prompt to run the commands. To install anaconda check this URL https://www.anaconda.com/download/ You will also need to download and install the below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.6 then run the below commands in command prompt/terminal to install these packages pip install -U sci-kit-learn pip install NumPy pip install scipy if you have chosen to install anaconda then run the below commands in anaconda prompt to install these packages conda install -c sci-kit-learn conda install -c anaconda numpy conda install -c anaconda scipy.

Dataset Used - Yale Faces Dataset

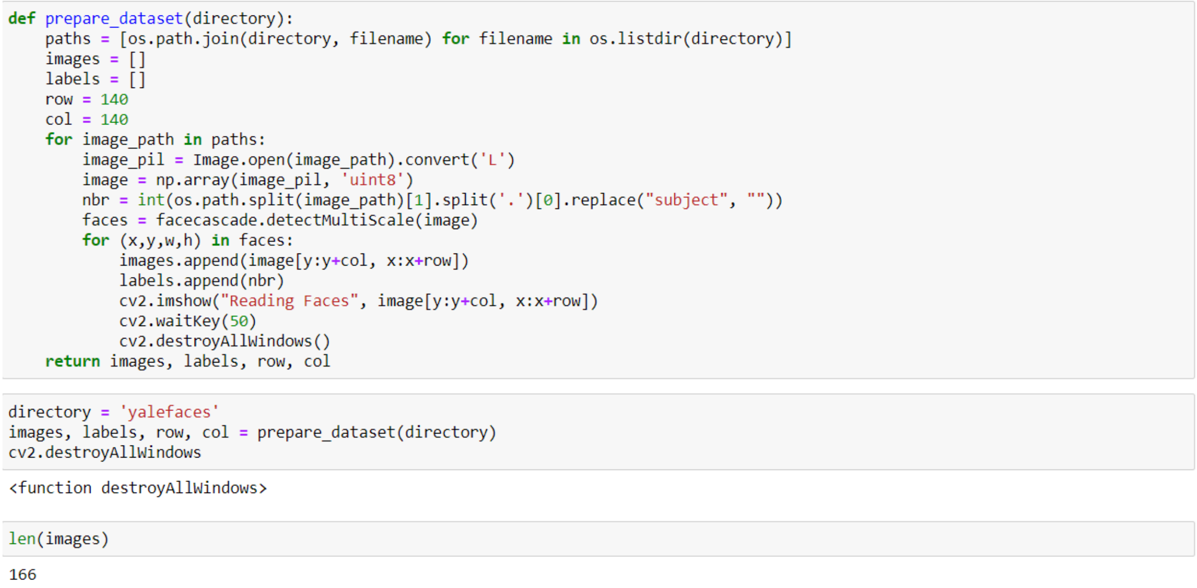
1. Importing libraries and Haarcascade Classifier



```
import cv2, os
import numpy as np
from PIL import Image
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

```
cascadelocation = "haarcascade_frontalface_default.xml"
facecascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

2. Preparing the dataset (Importing Images, Labels and resizing the images) -



```
def prepare_dataset(directory):
    paths = [os.path.join(directory, filename) for filename in os.listdir(directory)]
    images = []
    labels = []
    row = 140
    col = 140
    for image_path in paths:
        image_pil = Image.open(image_path).convert('L')
        image = np.array(image_pil, 'uint8')
        nbr = int(os.path.split(image_path)[1].split('.')[0].replace("subject", ""))
        faces = facecascade.detectMultiScale(image)
        for (x,y,w,h) in faces:
            images.append(image[y:y+col, x:x+row])
            labels.append(nbr)
            cv2.imshow("Reading Faces", image[y:y+col, x:x+row])
            cv2.waitKey(50)
            cv2.destroyAllWindows()
    return images, labels, row, col
```

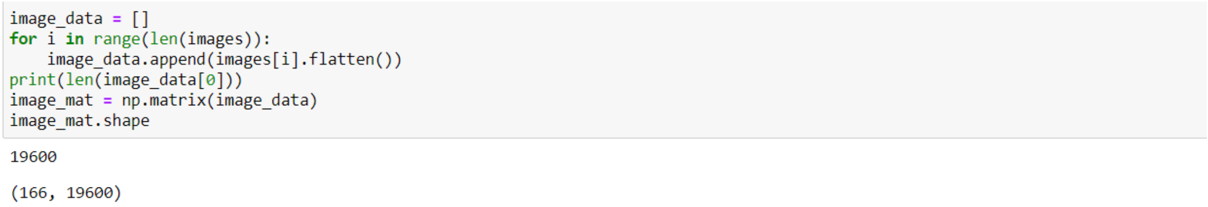
```
directory = 'yalefaces'
images, labels, row, col = prepare_dataset(directory)
cv2.destroyAllWindows
```

```
<function destroyAllWindows>
```

```
len(images)
```

```
166
```

3. Vectorising the Images (Flatten) -

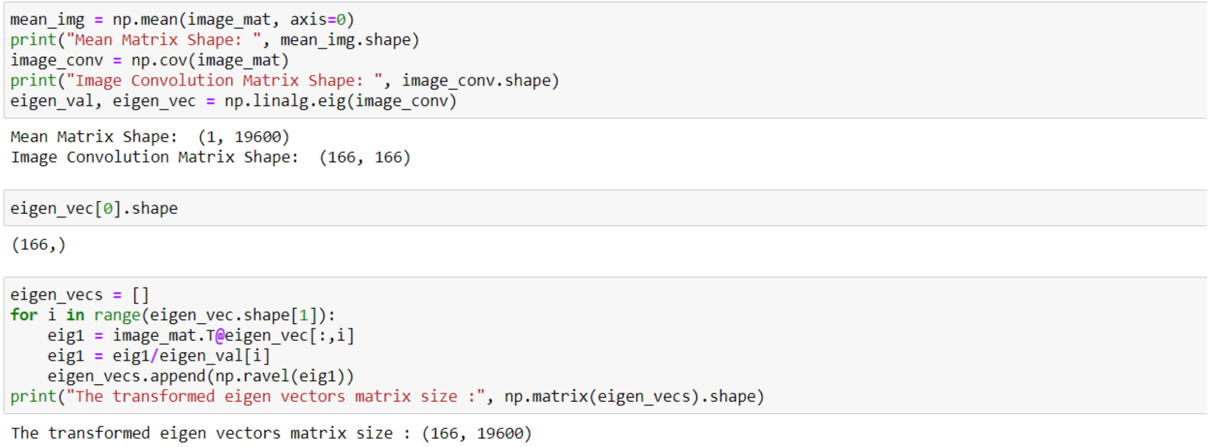


```
image_data = []
for i in range(len(images)):
    image_data.append(images[i].flatten())
print(len(image_data[0]))
image_mat = np.matrix(image_data)
image_mat.shape
```

```
19600
```

```
(166, 19600)
```

4. Dimensionality Reduction (PCA) -



```
mean_img = np.mean(image_mat, axis=0)
print("Mean Matrix Shape: ", mean_img.shape)
image_conv = np.cov(image_mat)
print("Image Convolution Matrix Shape: ", image_conv.shape)
eigen_val, eigen_vec = np.linalg.eig(image_conv)
```

```
Mean Matrix Shape:  (1, 19600)
Image Convolution Matrix Shape:  (166, 166)
```

```
eigen_vec[0].shape
```

```
(166,)
```

```
eigen_vecs = []
for i in range(eigen_vec.shape[1]):
    eig1 = image_mat.T@eigen_vec[:,i]
    eig1 = eig1/eigen_val[i]
    eigen_vecs.append(np.ravel(eig1))
print("The transformed eigen vectors matrix size :", np.matrix(eigen_vecs).shape)
```

```
The transformed eigen vectors matrix size : (166, 19600)
```

```
sort_ind = np.argsort(eigen_val)
sort_ind = sort_ind[::-1]

eig_val_sum = np.sum(eigen_val)

temp_sum = 0
principal_eig_vec = []
principal_eig_val = []
i = 0
while(temp_sum < 0.98*eig_val_sum):
    principal_eig_vec.append(eigen_vecs[sort_ind[i]])
    principal_eig_val.append(eigen_val[sort_ind[i]])
    temp_sum += eigen_val[sort_ind[i]]
    i+=1
print("Number of components is {}".format(i))
```

```
Number of components is 83
```

```
Q_hat = np.matrix(principal_eig_vec)
print(Q_hat.shape)
```

```
(83, 19600)
```

```
trans_image_data = image_mat@Q_hat.T
print(trans_image_data.shape)
```

```
(166, 83)
```

5. Approximate nearest search to find nearest neighbours (BallTree) on a test Image-

```
from sklearn.neighbors import BallTree
```

```
face_tree = BallTree(trans_image_data, leaf_size = 15)
```

```
test_img = trans_image_data[1:2,:]
test_img.shape
```

```
(1, 83)
```

```
f_dist, f_ind = face_tree.query(np.asarray(test_img), k=10)
```

```
f_ind
```

```
array([[ 1,  2, 56,  9,  5,  6, 10,  8, 11, 12]], dtype=int64)
```

```
f_dist
```

```
array([[    0.        ,     0.        , 18638.87599507, 19752.84248421,
        19891.85428494, 19900.22801594, 23157.35033651, 24056.68825821,
        27293.8301216 , 36647.89677826]])
```

```
for i, ind in enumerate(f_ind[0]):
    cv2.imshow("result", images[ind])
    if cv2.waitKey(0) == 'q':
        cv2.destroyAllWindows
cv2.destroyAllWindows
```

```
<function destroyAllWindows>
```

```
fig = plt.figure()
cols = 2
for n in f_ind[0]:
    plt.subplot()
    plt.gray()
    plt.imshow(images[n])
    plt.show()
plt.show()
```