

# Project Report

## Querying In Face Dataset

Name:Keshav Anand

Course: AI and ML

(Batch-4)

Duration: 12 months

Problem Statement: Querying in face dataset using Hashing technique.

### Prerequisites

---

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

Python 3.6 This setup requires that your machine has latest version of python. The following url <https://www.python.org/downloads/> can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: <https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/>. Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic.

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 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 below commands in command prompt/terminal to install these packages `pip install -U scikit-learn` `pip install numpy` `pip install scipy` if you have chosen to install anaconda then run below commands in anaconda prompt to install these packages `conda install -c scikit-learn` `conda install -c anaconda numpy` `conda install -c anaconda scipy`

### Dataset used:

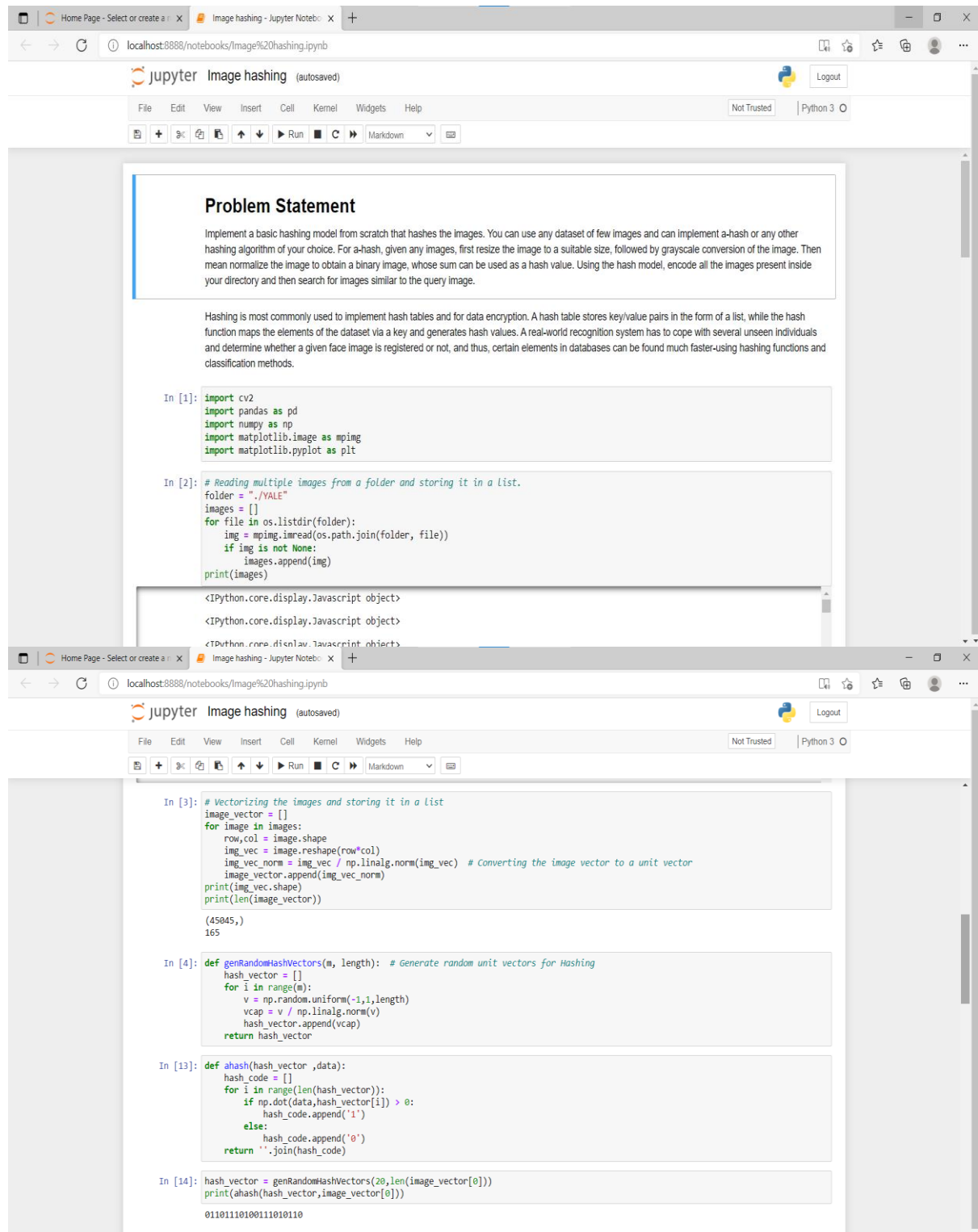
---

The dataset used is YALE-face dataset which is available on a website called Kaggle.com.

Method used for detection:

## HASHING

Screenshots of Source Code and Output:



The image displays two screenshots of a Jupyter Notebook titled "Image hashing" (autosaved). The notebook is running on a local host at localhost:8888/notebooks/Image%20hashing.ipynb. The interface shows the Jupyter logo, the title "Image hashing", and a "Logout" button. The notebook is divided into cells for code and output.

**Problem Statement**

Implement a basic hashing model from scratch that hashes the images. You can use any dataset of few images and can implement a-hash or any other hashing algorithm of your choice. For a-hash, given any images, first resize the image to a suitable size, followed by grayscale conversion of the image. Then mean normalize the image to obtain a binary image, whose sum can be used as a hash value. Using the hash model, encode all the images present inside your directory and then search for images similar to the query image.

Hashing is most commonly used to implement hash tables and for data encryption. A hash table stores key/value pairs in the form of a list, while the hash function maps the elements of the dataset via a key and generates hash values. A real-world recognition system has to cope with several unseen individuals and determine whether a given face image is registered or not, and thus, certain elements in databases can be found much faster using hashing functions and classification methods.

**Code and Output:**

```
In [1]: import cv2
import pandas as pd
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

In [2]: # Reading multiple images from a folder and storing it in a list.
folder = "./VALE"
images = []
for file in os.listdir(folder):
    img = mpimg.imread(os.path.join(folder, file))
    if img is not None:
        images.append(img)
print(images)
```

<IPython.core.display.Javascript object>  
<IPython.core.display.Javascript object>  
<IPython.core.display.Javascript object>

```
In [3]: # Vectorizing the images and storing it in a list
image_vector = []
for image in images:
    row,col = image.shape
    img_vec = image.reshape(row*col)
    img_vec_norm = img_vec / np.linalg.norm(img_vec) # Converting the image vector to a unit vector
    image_vector.append(img_vec_norm)
print(img_vec.shape)
print(len(image_vector))

(45045,)
165

In [4]: def genRandomHashVectors(m, length): # Generate random unit vectors for Hashing
    hash_vector = []
    for i in range(m):
        v = np.random.uniform(-1,1,length)
        vcap = v / np.linalg.norm(v)
        hash_vector.append(vcap)
    return hash_vector

In [13]: def ahash(hash_vector ,data):
    hash_code = []
    for i in range(len(hash_vector)):
        if np.dot(data,hash_vector[i]) > 0:
            hash_code.append('1')
        else:
            hash_code.append('0')
    return ''.join(hash_code)

In [14]: hash_vector = genRandomHashVectors(20,len(image_vector[0]))
print(ahash(hash_vector,image_vector[0]))

011011101001111010110
```

Home Page - Select or create a notebook | Image hashing - Jupyter Notebook | +

localhost:8888/notebooks/Image%20hashing.ipynb

Jupyter Image hashing (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

0110110100111010110

```
In [15]: # Creating a Image Dictionary using the hash as the keys
img_dict = {}
for i in range(len(image_vector)):
    hash_code = ahash(hash_vector, image_vector[i])
    if hash_code not in img_dict.keys():
        img_dict[hash_code] = [i]
    else:
        img_dict[hash_code].append(i)
```

```
In [16]: keys = list(img_dict.keys())
values = list(img_dict.values())
```

```
In [17]: print(img_dict)
```


```
{'0110110100111010110': [0, 88, 118, 159], '01101100000111010110': [1, 2, 34, 37, 45, 46, 48, 49, 51, 52, 53, 54, 59, 60, 62, 77, 81, 86, 101, 103, 105, 106, 107, 121, 122, 125, 126, 127, 130, 152, 155, 163], '0110110100111010110': [3, 102], '01101100011010110': [4, 5, 7, 8, 9, 10, 17, 31, 33, 35, 38, 40, 41, 42, 57, 67, 74, 75, 87, 99, 100, 108, 132, 141, 145, 154, 158, 162], '010011010011001011': [6], '01101100100111010110': [11, 153], '01101100000111010011': [12, 13, 18, 23], '0110110100111010011': [14], '01101101000111010110': [15, 47, 79, 89, 124, 129, 131], '01101100000111010111': [16, 20, 61, 64, 104, 144], '01101100000111010010': [19], '01101101000111010011': [21], '0110110100111010111': [22, 147, 151, 160, 164], '01101110000111010111': [24, 30, 43, 109, 143, 148, 150], '0010110101111010011': [25], '01101110000111010010': [26, 27, 29, 70, 71, 73, 136, 137], '11001110010111010011': [28], '01101110100111010011': [32, 161], '0110110100111010110': [36, 91], '01101110000010010111': [39, 149], '01101100000101010110': [44], '11101110000010010110': [50], '01101100000110010110': [55], '00101100000111010110': [56, 65], '0110110101111010110': [58], '00101110000111010110': [63], '01101110000101010110': [66, 68], '01101101001101010011': [69], '01101110000101010111': [72], '01101111000111010011': [76], '01101101000111010111': [78], '01101101100111010111': [80], '01101111000111010111': [82, 95, 96], '11101111000111010111': [83], '01101111100111010110': [84, 140], '0110111100110010110': [85], '11101100000111010110': [90, 97, 123], '0110111000111010110': [92, 93, 98], '11101110000111010111': [94], '0110111010111000110': [110], '01101110001111010110': [111], '01101110001111011111': [112], '01101100101111011110': [113], '0110111000111011110': [114], '01101110001111010111': [115], '01101100000110000111': [116], '00101110001111010110': [117], '0010111000010011110': [119], '01101110100110010110': [120], '01101101000110010110': [128], '01101110000111010011': [133, 134, 139, 142], '011011110101010011': [135], '01001110010111010010': [138], '0110110101110010111': [146], '01101110100111010010': [156], '0110110101111010111': [157]}
```

In [18]: # Plotting images with same hash code

```
def plotImages(images, img_indices):
    imgs = [images[i] for i in range(len(images)) if i in img_indices]
    fig = plt.figure()
    cols = 2
    n_images = len(imgs)
    for n, image in zip(range(n_images), imgs):
        ax = fig.add_subplot(cols, np.ceil(n_images/float(cols)), n + 1)
        plt.gray()
        plt.imshow(image)
        fig.set_size_inches(np.array(fig.get_size_inches()) * n_images)
    plt.show()
```

```
In [22]: plotImages(images, values[1])
```

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\_launcher.py:8: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.



Home Page - Select or create a notebook | Image hashing - Jupyter Notebook | +

localhost:8888/notebooks/Image%20hashing.ipynb

Jupyter Image hashing (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

111': [157]}

```
In [18]: # Plotting images with same hash code
def plotImages(images, img_indices):
    imgs = [images[i] for i in range(len(images)) if i in img_indices]
    fig = plt.figure()
    cols = 2
    n_images = len(imgs)
    for n, image in zip(range(n_images), imgs):
        ax = fig.add_subplot(cols, np.ceil(n_images/float(cols)), n + 1)
        plt.gray()
        plt.imshow(image)
        fig.set_size_inches(np.array(fig.get_size_inches()) * n_images)
    plt.show()
```

```
In [22]: plotImages(images, values[1])
```

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\_launcher.py:8: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.

